KTH Computer Science
and Communication

# A real time performance system for interactively layered audio sequences

Leonardo Aldrey

Supervisor: Prof. Sten Ternström

Approved on …… Examiner: ……
(printed name)

KTH – School of Computer Science and Communication (CSC)
Department of Speech, Music and Hearing
100 44 Stockholm

CONTENTS

**Abstract**

**I. Introduction ..... 1**
Motivation ..... 1
Other systems ..... 2
Loop Machines ..... 2
Sequencers ..... 5
Live tools ..... 6
Discussions with artist ..... 8
Juana Molina (Argentina) ..... 8
Todos Santos (Venezuela) ..... 9

**II. Music Theory Aspects ..... 10**
Musical structure ..... 10

**III. Design considerations ..... 11**
Choice of programming platform ..... 11
Choice of Hardware ..... 11
Usability ..... 13
Need for innovation ..... 13

**IV. Description of the system ..... 15**
Overall structure ..... 15
Scene navigator blocks and description ..... 16
1. Scene's memory ..... 17
2. Clock gate ..... 20
3. Internal Status ..... 22
4. Next State initialization ..... 24
5. Sample pointer system ..... 25
Audio tracks ..... 28

**V. Work in progress ..... 30**
Things remaining to be designed or implemented ..... 30
Optimization possibilities ..... 31

**VI. Development history ..... 32**
Performance needs ..... 32
Other systems analysis ..... 32
Learning about Reaktor and its possibilities ..... 32
No audio sequencers ..... 32
Approaches rejected ..... 33
Approaches used ..... 33
Rec, Overdub, Playback, Transfer and Do nothing modes ..... 33
Approaches rejected ..... 34
Approaches used ..... 34
Managing scene lengths ..... 34
Diary of the project and info in the structure of the program ..... 35

**VII. Quality control tests and results ..... 36**

**VIII. Appendix ..... 40**
Terminology ..... 40

**IX. User Manual ..... 41**

## Abstract

**Background:** The cost is decreasing for including electronic equipment in modern styles of musical performance. This has opened up a wide variety of possibilities for creating new forms of performing live music. Some multi-instrumentalist musicians have developed techniques to record layers of audio, during their own performance to use them as background tracks. A computer based platform was designed that is capable of creating a "map" or "template" of the song that the musician will use to record background tracks while performing. **Results:** Scene Builder is a system capable of storing the structure and flow of layers in a song, in order to allow performers to focus on their instruments and reproduced live their music creation process. Songs are treated as a sequence of scenes, programmed individually, that are triggered by a MIDI foot controller.



**Keywords:** performance, scenes, audio layers, multi-track live recording.

# I. Introduction

## Motivation

The role of computers in current musical styles has become very important. From composition, to performance they represent a wonderful tool that has expanded the musical world to an endless number of possibilities. In musical performance, computers are used for many purposes such as sound synthesis, sequencing, processing effects, sampling, playback of background tracks, among others. Nevertheless in some cases, the role of computers has taken prominence from the performer, producing a lack of musicality and less "human interaction" with the audience.

In the quest for balance between the benefits of technology and musicality, different trends of musical performance have emerged. We are particularly interested in the performance styles where the musicians try to minimize the use of prerecorded background tracks by creating live the maximum amount of elements. As a reference to analyze this type of performance we have chosen Argentine singer/songwriter Juana Molina. In her presentations, all background tracks are produced live with a set of loop machines and effects. This technique allows her to create and combine different layers, building a background track to play along with.

This performance style combines technology, creativity and musicality to create a very interesting communication process, where the audience is able to hear and see a reproduction of the music creation process, recognizing all the elements that make up the song as they appear in the performance.

Nevertheless, contemporary loop machines have certain limitations, such as the lack of flexibility and the absence of procedures to organize musical structures. To really exploit the possibilities of an advanced loop machine (Such as Boss RC-50) there are many configurations that have to be done during the performance, and this will make the song less fluent. In most cases to configure the advanced functions of the system it is required to manipulate dials, use small buttons to access different modes of the foot controller and other procedures where the artist has to stop performing in order to prepare the system.

In this Master Thesis, the goal was to design and implement a prototype that expands the possibilities of a loop machine in order to facilitate performances that are more musical.

## Other systems

The following sections describe briefly the characteristics of existing systems. They are of special interest for the later section of the report where the needs for innovation are explained.

## Loop Machines

### Stand-alone machines (Boss RC-50)

**Figure 1** Boss RC-50 loop station

This loop machine is considered to be one of the best loop machines available in the market. It consists of a pedal controller with built-in software to work as standalone equipment.

The RC-50 loop station has seven pedals. Three of these pedals are used to select the phrases, and the other four to trigger and control the behavior of the phrases (Rec/Play/Overdub, Stop, Tempo, and Redo/Undo). In addition to this there are several small buttons and dials that are used to configure the other parameters of the system. For more detailed description the manuals are available in their web site[1].

A brief summary of the major features of the RC-50 is given below:

**USING SINGLE MODE**

In single mode only one phrase can be played at a time. This mode is good in situations where the user wants to record and play phrases in a sequential order. This mode requires some settings.

**1. Phrase change mode (Immediate, loop end)**

To determine if the selected phrases will be played immediately or if the system will wait until the loop ends.

[1] http://www.bossus.com/index.asp?pg=1&tmp=155

### 2. Loop or one-shot playback

It allows you to decide if you want to play a phrase only once or if you want to loop it.

### 3. Background prerecorded samples.

There is a built-in guide that provides background rhythmic to play along.

### 4. Tempo sync

This option causes a previously recorded phrase to be played at the current patch tempo.

### 5. Pedal operation in single mode

Description of the function of each pedal, depending on the configuration of each phrase.

The pedals are:

- Rec/Play/Overdub
- Stop
- Tempo
- Undo/Redo
- Phrase1
- Phrase2
- Phrase3

## PATCHES

This is where most of the configuration of the system is done.

A patch contains the following elements:

- Three phrases (3 clips, one per channel)
- Various phrase playback settings.
- Guide settings (settings for the background tracks)
- Real-time control assignments for all the foot controllers.
- Master tempo setting (Used by tempo sync and quantize)

### Patch Settings

1. **Play mode:** includes the option of single or multi mode, to decide whether the phrases will be played simultaneously or not. Also the "loop sync" button is used to synchronize phrases with each other.
2. **Auto recording:** When this feature is enabled, the system does not begin until it receives audio from one of the input jacks.
3. **Input mode:** mono or stereo.
4. **Overdub mode: Overdub** combines previous audio with the line in. **Replace** to substitute the previous audio. **Momentary punch in** to replace audio only while the REC/PLAY/OVERDUB button is pressed.

5. **Patch tempo:** tempo reference for a number of functions including the guide, tempo sync, and loop quantize.
6. **Guide settings:** to determine the use of background rhythms or audio samples.
7. **Patch name**
8. **Patch level**
9. **Patch phrase change** (immediate/loop end)
10. **Patch: fade in, fade out.**
11. **Simultaneous start:** allows the user to determine if tow or more phrases will be started at the same time.
12. **Patch inputs and outputs**
13. **Patch MIDI synch:** to configure how the system will respond to MIDI messages

This patches and how they are configured is shown in greater depth in the user manuals. Here, the interest is to mention the possibilities of the loop station and more importantly the needs for innovation that are described in the following section.

### Computer based machine (Guitar Rig)

**Figure 2** Loop Station by Native Instruments. Part of Guitar Rig 2

Guitar Rig 2 is a complete system to manipulate and transform audio, mainly designed for guitarists. It consists of software that has a set of tools and effects and a pedal controller that is used as a physical hardware interface to control the program during the performance. As a loop station, it is not as advanced as the Boss RC-50, but it has a structure (Computer program + foot controller) that is more similar to the implementation of this project.

For a more accessible overview of the program, the reader is referred to the "Loop Machine Tutorial Video" available on the Native Instruments web site[2].

[2] http://www.native-instruments.com/index.php?id=gr2tut3_us

**GENERAL FEATURES**

1. **As many layers as you want:** Since it is a computer program, the storage limit is determined by the computer, which means that there is a great deal of capacity in the system.
2. **Store a complete jam session:** Guitar rig affords storage of all the layers that you create during the jamming session with the loop machine.
3. **Pedal assignation:** The pedals of the foot controller can be assigned to different buttons and controls of the system e.g. record, stop, undo, redo, etc.
4. **Tempo sync:** it is used to synchronize the tempo of the loop and its length to a metronome.
5. **Different presents:** The presets and effects of guitar rig can be chosen for the different layers.
6. **Undo a delete features for each layer:** The program provides undo and redo of the last layer created. The user can undo repeatedly to erase layers one by one.
7. **Auto start for song and overdubbing:** The user can choose to trigger the start of the song with an audio input signal. Also when selecting overdub, this mode stays in standby until there is audio at the input.
8. **Loop A/B for two different phrases:** The program allows the creation of two different phrases or scenes. It has a button A/B to select in which one the user will be working and they can be used for example for chorus and verse.
9. **Export Layer and export mix:** it has the feature to export particular layers or the complete mix so they can be used in another sequencer or different applications.
10. **Load audio and presets:** The system allows you to load audio in the loop machine and also presets for a particular use of the loop machine.

## Sequencers

There are several sequencers in the market with features that are of interest for this project. Here, Nuendo 3 and Cubase were examined in order to understand the logic of the system and their possibilities.

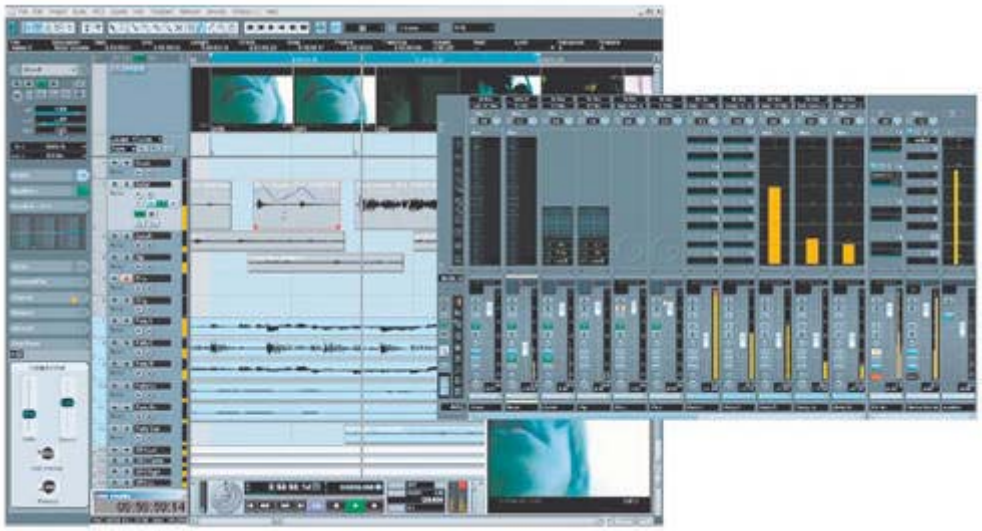


**Figure 3** Nuendo 3: Sequencer from Steinberg

There are many features[3] but among the most important ones are:

- Recording, editing and mixing with up to 192-kHz and 24-bit processing power (pristine 32-bit floating point internal resolution)
- Intuitive handling and extensive customization options to accelerate workflow
- Projects port readily to and from all standard audio and video cutting systems
- Powerful multi-channel capability throughout the signal path
- 32-bit Audio Engine optimized for multiprocessor operation
- Up to 192 physical inputs/outputs alongside freely selectable audio hardware
- Integration into LAN and WAN environments
- Cross-platform product (Mac OS X/Windows XP)

These programs are very complete but their design is not meant for live performances. Still, some principles were used in the design of the present project, mainly regarding user's choices for each track (input, outputs, effect routing, etc) and to design the graphical interface.

## Live tools

### Reaktor Live tools

In Reaktor there are a few "instruments" designed for live performance. They are mostly samplers that use prerecorded audio samples and then trigger and modify them with effects and other editing tools

In the user library there are some instruments that are able to record audio in a similar way to a loop machine, but none of them consider the possibility of creating sequences of such loops to create a song. Some of them have good ideas, but their implementation is very limited in the sense that the user has to be always controlling the software with the mouse, limited to one track; and their design is not meant for creating complete musical structures.

Among the best features of relevance to this project are:

## Live Sampler v1.0

*First version of a sampler designed for live use (Live Sampler by Red Wierenga)*

**Figure 4** Loop machine made in Reaktor

This is a sampler which holds up to eight 10-second samples (can be increased by increasing Audio Table size). You can set recording time in advance or on the fly. You may

[3] Information about Nuendo and Cubase, manuals and other tutorials in http://www.steinberg.net/89_1.html

set the sampler to, upon ceasing recording, commence playback, looping, overdubbing, reverse direction, or my favorite, the old "To and Fro."

### Scenario

Another very useful instrument found in Reaktor's library is "Scenario" which is a live tool mainly used as a sampler to trigger, play and transforms samples in a live performance. This instrument can be found in the User libraries[4].

**Figure 5** Scenario: Mixing instrument design in Reaktor.

The most important idea of the concept of this instrument that was used in the project was the translation between what I though as "States of the performance" to what they consider "Scenes". This is a very useful way to navigate the different elements that constitute the song.

[4] http://www.native-instruments.com/index.php?id=userlibrary_us&type=0&ulbr=1&plview=list

## Discussions with artist

## Juana Molina (Argentina)

**Figure 6 Juana Molina** (eminent multiinstrument artist who acted as reference for the project)

To discover some the limitations and advantages of the system based on a set of loop machines, several discussions via e-mail were held with Argentinean singer/songwriter Juana Molina[5]. The different issues mentioned by the artist were:

**Latency:** the artist expressed that some systems present latency (delay between the physical action and the system response). This produces failures in the consistency of the tempo and rhythm in the piece.

**Number of tracks:** Loop machines have a limited number of inputs. For the artist, four different channels are enough to create all the elements in her music.

**Preparation of equipment for each performance:** For every performance the loop machines have to be routed, together with mixers and effects. The system could be more efficient if it worked as a bundle with all the elements required.

**Loop access and their combination:** To switch among different loops recorded in the machine, it is required to press several buttons to address the loop that you want to play. A system to access more directly to the loops would be favorable.

**Simplicity of the pedal controller:** The artist expressed that the best system will be the one that maximizes the performance features with less pedal controllers. During the performance the idea is to focus on the musical part and reduce to the maximum the technical aspects.

[5] Juana Molina available in http://www.myspace.com/juanamolina

## Todos Santos (Venezuela)

Todos Santos[6] is a Venezuelan band in the genre of electro-acoustic music. Their performances combine computer elements with the use of the computer for creating beats, playback of background tracks and sound effects. The band works with a software called **Ableton Live** which is a sequencer that has some special features for live performance.

The discussions held with the band, were mainly to understand the features of the program as well as the limitations during performance.

Among the most important aspects of their performance was that one of the members of the band had to be controlling the software all the time. This implied the use of the mouse and the configuration of the different parts of the song, which required the performer to use the hands and therefore stop performing any instrument.

On the other hand they expressed that Ableton Live was a very powerful tool to trigger clips and sounds, as well to manipulate them in real time during the performance.

[6] Todos santos myspace
http://profile.myspace.com/index.cfm?fuseaction=user.viewprofile&friendid=34592491&MyToken=6e28b4a2-4710-4e0e-a33e-b7c34f3bb154

# II. Music Theory Aspects

## Musical structure

The system developed through this project is based on certain principles of musical structures that are commonly applied in popular music. Some of these concepts are briefly mentioned below, for their importance in the implementation of the system.

### Sections

Most popular music is based on a sectional form. The sections are usually **Verse, Chorus, Bridge and Vocal runs** (small variations of the chorus). The fact that most popular music is based on this structure tells us that **repetition** is a very important element of music; hence a system that is able to repeat elements of the song could be very useful during performance.

### Melody, Harmony and rhythm

These elements are said to be some of the most important ones in popular music. For the design of the system t was important to consider that there should be a way to allow the musician to create all this elements during the performance. To be able to create and combine all this elements, multiple tracks should be available in order to perform different instruments (percussion, keyboards, guitar, voice, etc)

### Continuity

In many musical styles continuity is an important aspect. In the evolution of the song, there are usually some elements that remain unchanged, and others that are added or modified. This principle is very important for the use of the system. The musician has to create the songs knowing the performance limitations (performing probably one instrument at a time) and also the flow of the elements that will be combined to create the song.

### Musical forms (Rondo)

Rondo is a musical form that is characterized by "returning" to the main theme of the piece. There is a principal theme usually called "Refrain" that is alternated with contrasting themes called episodes. This concept is also interesting to understand some possibilities of musical forms that could be explored with the system designed in the project.

# III. Design considerations

## Choice of programming platform

Several platforms were considered for programming the "Scene Builder". Among them were Pure Data, MAX/MSP, and Reaktor. Each had its advantages and disadvantages. Reaktor was chosen for the following reasons:

**Flexible programming environment:** Reaktor has the advantage that it has many ready-to-use instruments, effects, samplers, sequencers, etc.; and it is possible to open them and inspect how they were programmed. This gives the programmer the possibility to study other designs and incorporate or modify elements to meet specific needs.
**Huge library:** The program comes with a large library of instruments. These instruments include samplers, sequencers, effects, etc. Using this program to develop the project will open the possibility of expanding the system by adding all this different instruments.
**Graphical Interface:** one of the most important advantages of Reaktor is that is very practical for creating the user graphical interface. There are many possibilities to design the interaction with the user. The program has different views, which allow separating the programming stage with the performing stage. This is done directly by a button in the panel of the program that switches between "Instrument Structure" and "Instrument panel".
**Control of parameters:** an important part of the concept of the project is to be able to configure each scene in a particular way (including all the effects and plug-ins used). Scene Builder had to be able to communicate in an easy way with this additional elements used during the performance. Having "Scene Builder" and the effects in the same platform makes easier to combine them to perform together.
**Forum and support:** Reaktor's forum is good; there are users around the world developing instruments and sharing their knowledge. This also opens the possibility of interacting with other programmers that could improve the project.
**Original Instruments:** there are many instruments that could be easily used with Scene builder during the performance. Among this are synthesizers, sound generators, surround utilities, sequencers, samplers etc.
**Very good sound quality:** Reaktor has shown to be able to produce a very good sound quality with a good computer. This was also very important to guaranty that the audio can be stored and played with good quality.

## Choice of Hardware

There are different pedal controllers that could be used for the implementation of the project. The idea of the software is to reduce as much as possible the manipulation of the foot controller by pre-programming the system with the decisions to make when pressing some pedal controller. Therefore, the controller used with "Scene Builder" can be very simple with the minimum requirement of having at least 7 pedals.

To implement this project, the Behringer FCB1010 was chosen because it has many features that can be used to expand and improve the project. For example this foot controller has two expression pedals that could be assigned to the effects used for the tracks. Due to the design of "Scene builder", these pedals could be used for a different effect or parameter in each scene, making the use of the pedal more diverse.

**Figure 7** Pedal controller for the hardware part of the project

### Features

- 10 banks with 10 user-editable presets each
- 2 expression pedals with freely assignable MIDI channel, controller number, and range
- 5 MIDI program change and 2 MIDI controller commands per preset (simultaneously transmittable)
- MIDI note-on commands transmittable for trigger and tap-tempo applications
- 2 programmable relay-controlled switch jacks, e.g. for amp channel selection
- MIDI merge and sysex backup functions
- Easily programmable in a standing position
- Integrated power supply -- no AC adapter, includes power cable

## Usability

"Scene Builder" is designed for the multi-instrumentalist musician who desires to perform songs by recording clips from different instruments and distributing them as sound layers to create a song. It could also be used as an advanced loop machine to perform with a band, but with the consideration that a machine will be playing audio so the musicians have to synchronize with it.

**The artist process: from composition to performance.**

**Steps:**

1. **Create a composition** thinking of the layers and flow in your musical performance.

2. **Division of the performance in scenes**.

3. **Configure each scene** of the performance:
    - Define each track's **RPO evolution**. The RPO evolution (Record-Play-Overdub) represents how the channel's mode will evolve within the scene when the user presses the RPO button (Record, Playback, Overdub, Transfer, Do nothing).
    - Define **audio transfers**. Each scene is capable of transferring audio from a previous scene. The user can determine if audio will be transferred, and from which previous scene it will come from.
    - Configuration of **volume levels and effects**. For each channel/scene the user can determine the input volume, the feedback volume (in case overdub is going to be used), configure the global mixer in a different way, and the effects used for each channel.
    - **Write in the event table** the values established in step three. This is done by pressing the write button.

4. **Practice** the performance and make **final adjustments**.

5. **Perform,** filling the created template with the audio.

## Need for innovation

The needs to be fulfilled by the system include all the normal features of a loop machine. Some of these features are:

- Good sound quality
- Low latency
- Be able to Record, Overdub and Playback.

The system designed through the project should be able also to improve some aspects that are enumerated below.

**1. Patch storage capability.** All the configurations needed to perform a scene are stored in the system. During the performance the user only has to recall them. This saves

time in configuring the system during the performance. Equalization, channel mode and evolution (Playback, Record, Overdub, Transfer, Do nothing), mixer configurations and other effects are stored for each scene**.**

**2. Maximum parameter control with fewer pedals.** Since the system is able to create a template of the configuration of the system along the song, the pedals are use only to control the transitions, while all the changes in the system are automatic during the performance.

**3. Distribution of loops along the song**. The program allows audio transfers to future parts (scenes) of the song, which allow the musician to reuse loops and samples in order to avoid repetition and therefore make a more fluent performance.

**4. Recombination of scenes.** Due to the structure of the program, new stages can be created by the combination of the content of different previous scenes. For example, in scene 4, the background track could be formed by: Channel 1 scene 2, Channel 2 scene 3, Channel 3 scene 1 while recording new audio on Channel 4.

**5. No need to stop the song for configurations.** In a normal loop machine, some configuration changes require to stop the performance. This is not necessary because the storage system predetermines these changes. The user only has to change scene without stopping the performance.

**6. Easy pedal manipulation.** The complexity of normal loop machines is absorbed by the capabilities of the system to store the configuration of each loop.

**7. Better graphical interface to visualize status of the system.** The normal loop stations have a very small screen with some symbols that are difficult to visualize and in some cases not very intuitive to learn.

# IV. Description of the system

## Overall structure

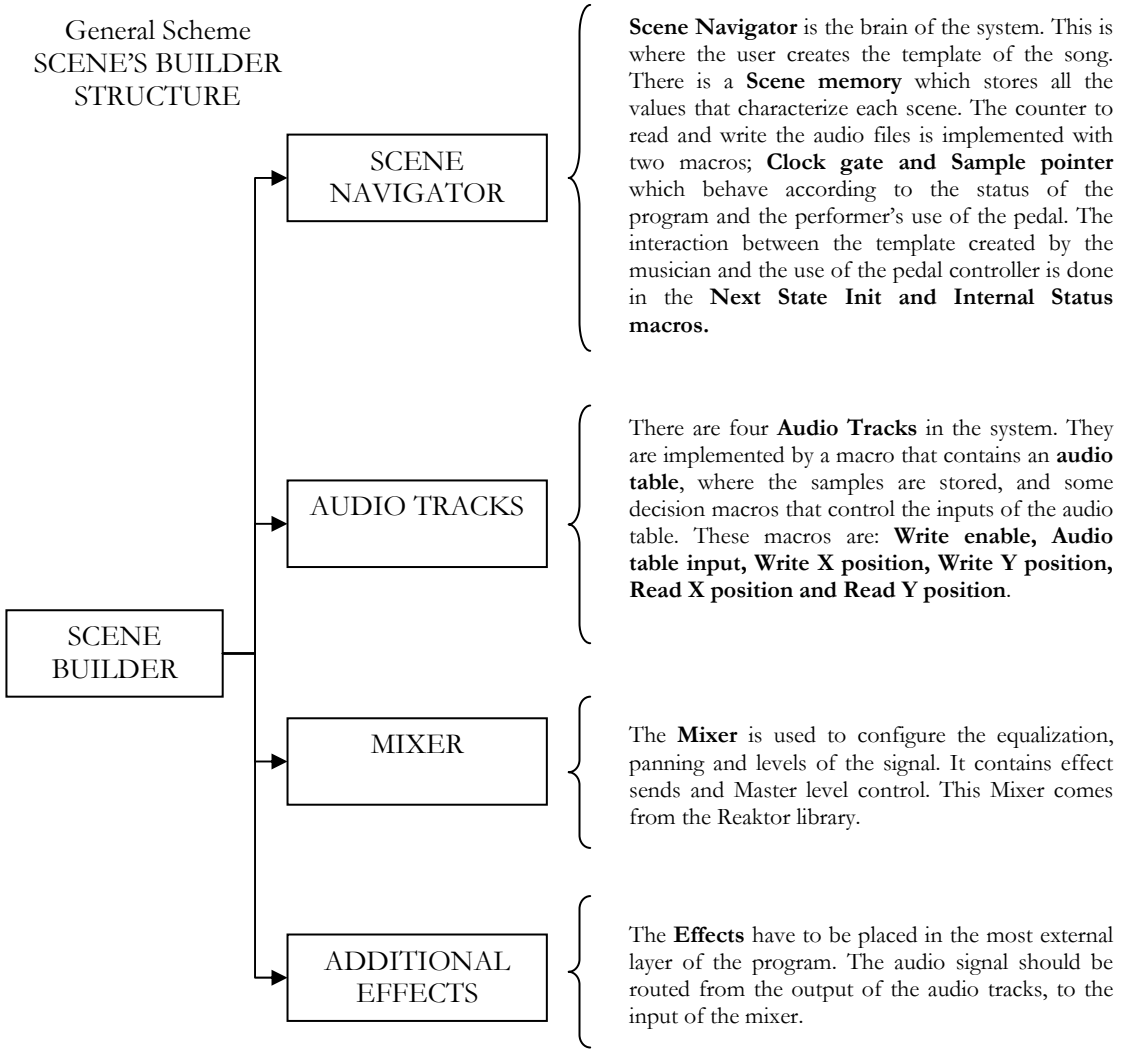

General Scheme
SCENE'S BUILDER
STRUCTURE
SCENE BUILDER
SCENE NAVIGATOR
AUDIO TRACKS
MIXER
ADDITIONAL EFFECTS
Scene Navigator is the brain of the system. This is where the user creates the template of the song. There is a Scene memory which stores all the values that characterize each scene. The counter to read and write the audio files is implemented with two macros; Clock gate and Sample pointer which behave according to the status of the program and the performer's use of the pedal. The interaction between the template created by the musician and the use of the pedal controller is done in the Next State Init and Internal Status macros.
There are four Audio Tracks in the system. They are implemented by a macro that contains an audio table, where the samples are stored, and some decision macros that control the inputs of the audio table. These macros are: Write enable, Audio table input, Write X position, Write Y position, Read X position and Read Y position.
The Mixer is used to configure the equalization, panning and levels of the signal. It contains effect sends and Master level control. This Mixer comes from the Reaktor library.
The Effects have to be placed in the most external layer of the program. The audio signal should be routed from the output of the audio tracks, to the input of the mixer.

**Figure 8: General Structure of Scene Builder.** This are the main blocks that constitute Scene Builder. The first two components, Scene Navigator and Audio tracks are the ones designed in the project and the last two are taken from the Reaktor library.

## Scene navigator blocks and description

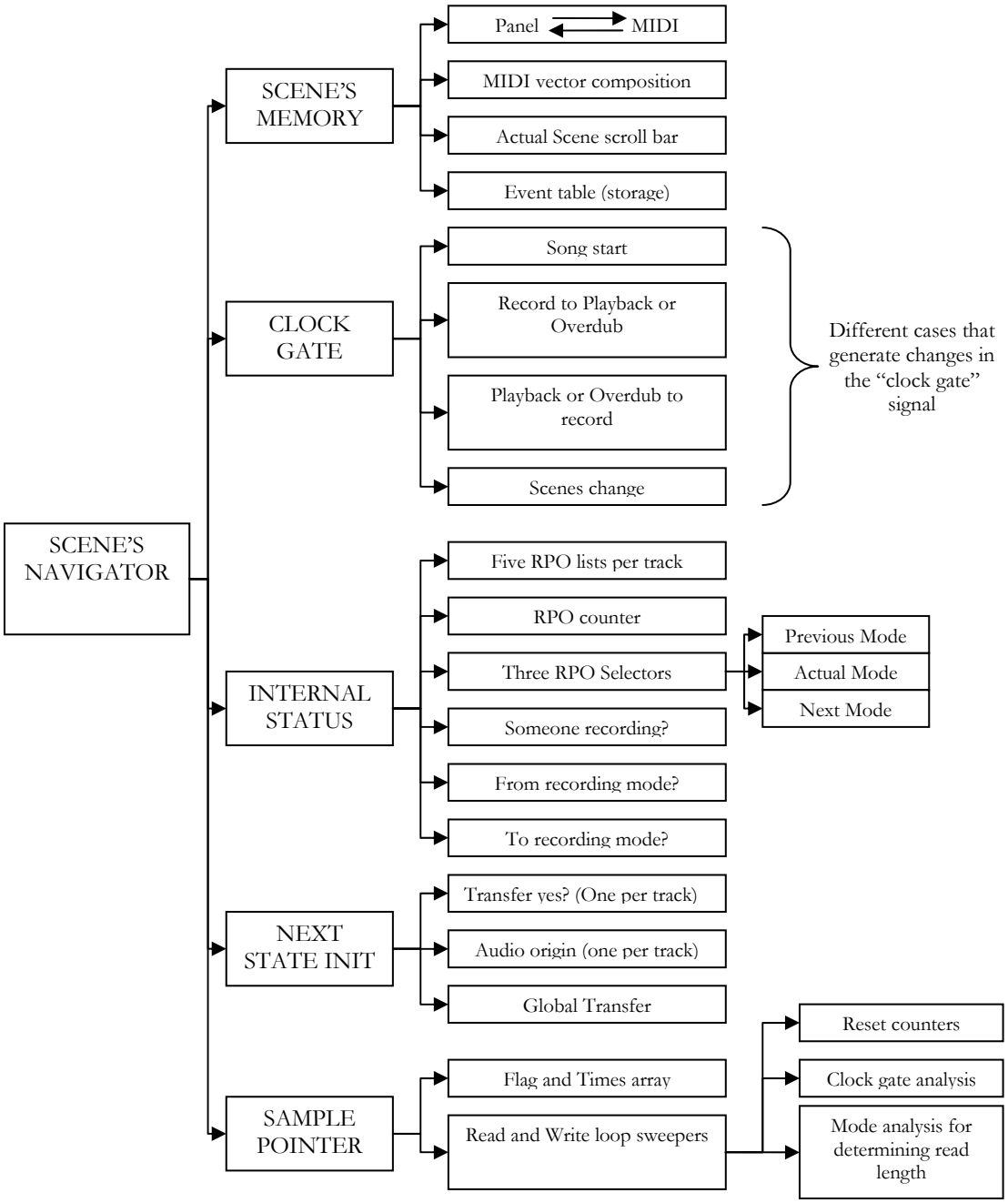



**Figure 9** Scene Builder's navigator general structure

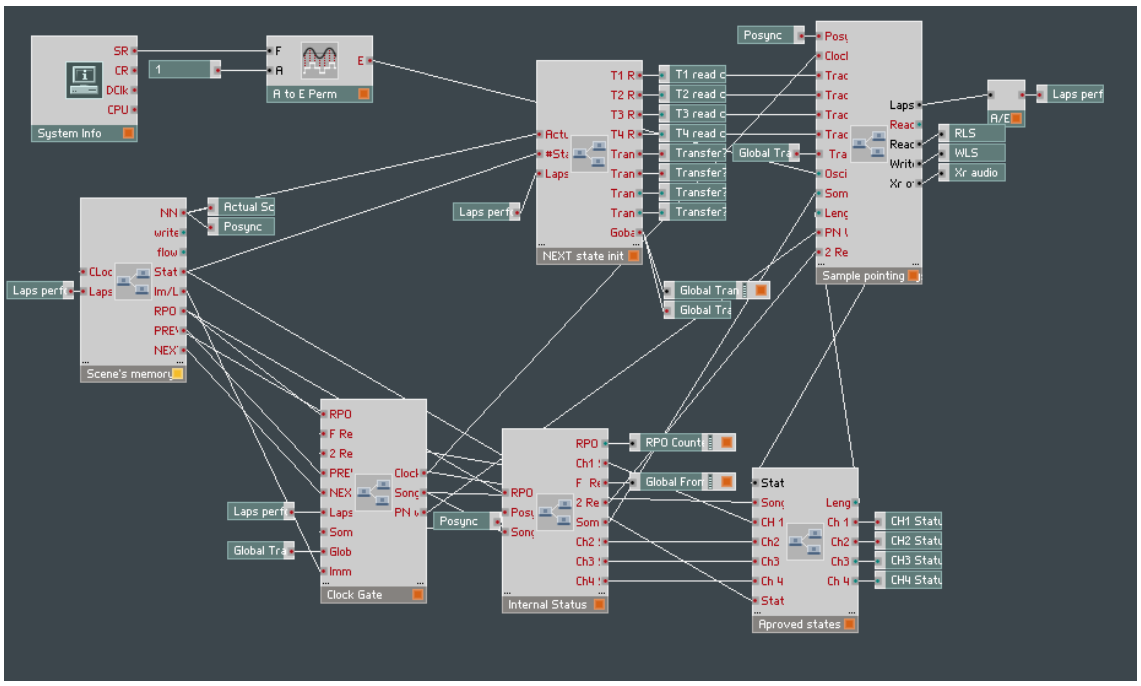



**Figure 10** Scene's Navigator Structure. Main blocks of the system

1. **Scene's memory**: This is the memory of the system, used to store the configuration of each stage. A graphical description of this macro is shown in Figure 11. The main blocks of this macro are given below.
   - **State navigator scroll bar:** graphical interface for showing the actual scene. The user can move the cursor to navigate the scenes of the song; this position is used to point the event table in order to change the configuration of the system. This scroll bar also is modified by the pedal **NEXT/PREV** buttons which are use to move along the song.
   - **Event Table:** It's the structure that stores the values of all the possible parameters to be configured in the system. It is organize as a matrix, where every row represents a different scene, and each column represents a parameter of the program. All controllers of the program are assigned to a column in the table and when the user presses the **"Write button"** these values are stored in the table. When the user moves along the scenes, the table is read and therefore all parameters are updated to the values stored.
   - **MIDI vector construction:** To link the table with all the parameters of the program, an internal MIDI vector is created. Each control in the system is assign to a MIDI control/note that is received by a MIDI input. All this MIDI notes are put together in a vector that feeds the table as a polyphonic voice. The output of the table is also routed to MIDI outs, in order to modify the controls in the program.
   - **Write, Unplanned, Flow logic:** Write button to tell the table when to store the values. Flow to allow changing states without making changes in the controls (useful to copy scene's parameters).
   - **Block: State Navigator.**

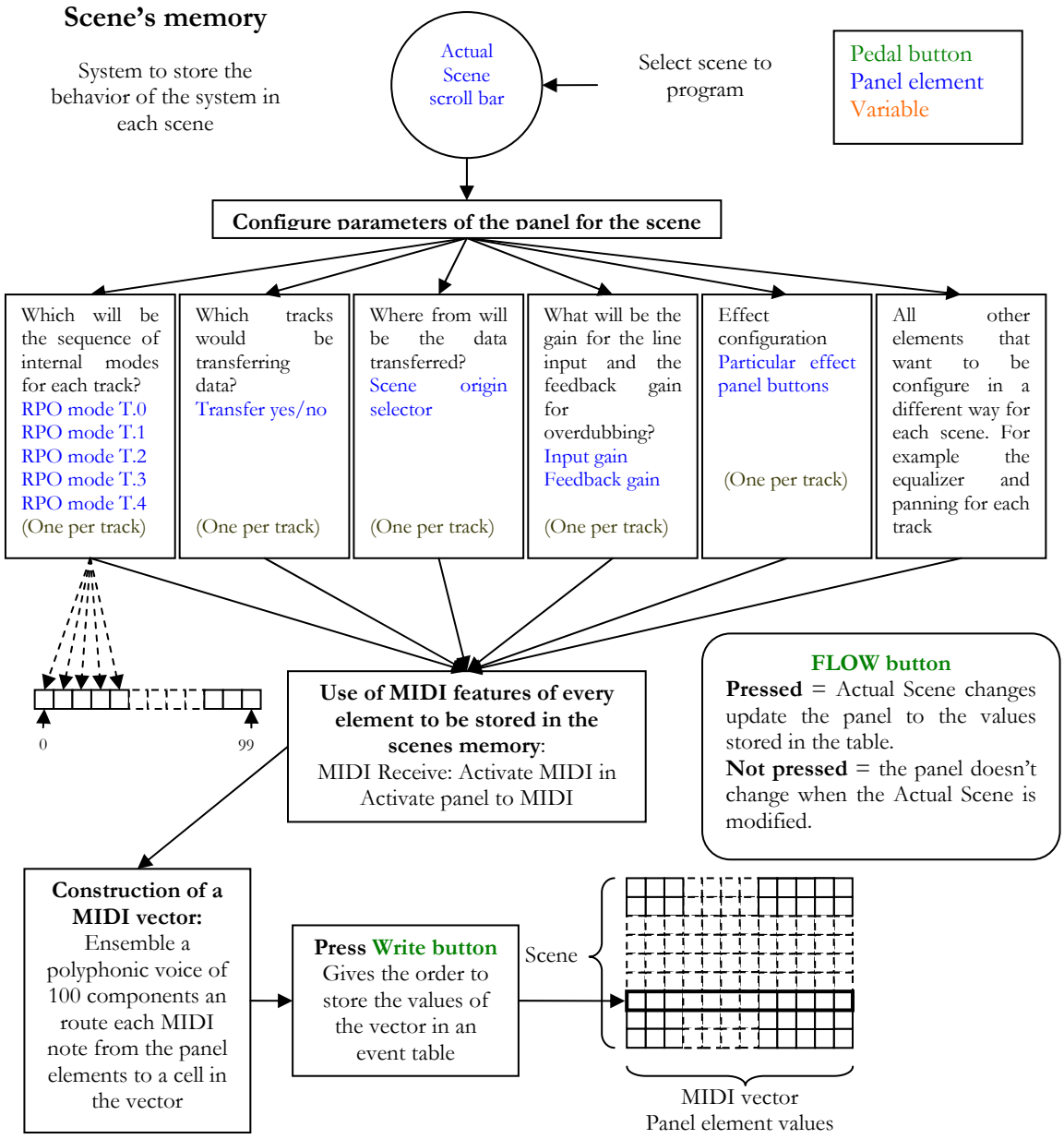

Scene's memory
System to store the behavior of the system in each scene
Actual Scene scroll bar
Select scene to program
Pedal button
Panel element
Variable
Configure parameters of the panel for the scene
Which will be the sequence of internal modes for each track?
RPO mode T.0
RPO mode T.1
RPO mode T.2
RPO mode T.3
RPO mode T.4
(One per track)
Which tracks would be transferring data?
Transfer yes/no
(One per track)
Where from will be the data transferred?
Scene origin selector
(One per track)
What will be the gain for the line input and the feedback gain for overdubbing?
Input gain
Feedback gain
(One per track)
Effect configuration
Particular effect panel buttons
(One per track)
All other elements that want to be configure in a different way for each scene. For example the equalizer and panning for each track
0
99
Use of MIDI features of every element to be stored in the scenes memory:
MIDI Receive: Activate MIDI in
Activate panel to MIDI
FLOW button
Pressed = Actual Scene changes update the panel to the values stored in the table.
Not pressed = the panel doesn't change when the Actual Scene is modified.
Construction of a MIDI vector:
Ensemble a polyphonic voice of 100 components an route each MIDI note from the panel elements to a cell in the vector
Press Write button
Gives the order to store the values of the vector in an event table
Scene
MIDI vector
Panel element values

**Figure 11 Scene's memory.** This diagram shows how the system stores the values of all the parameters that are configured during the programming stage of a song. The values of these parameters are stored in an event table. The rows represent the Scenes of the song and the columns the panel controls. For each Scene, all the parameters are bundled in a polyphonic voice that will fed as a row of the event table

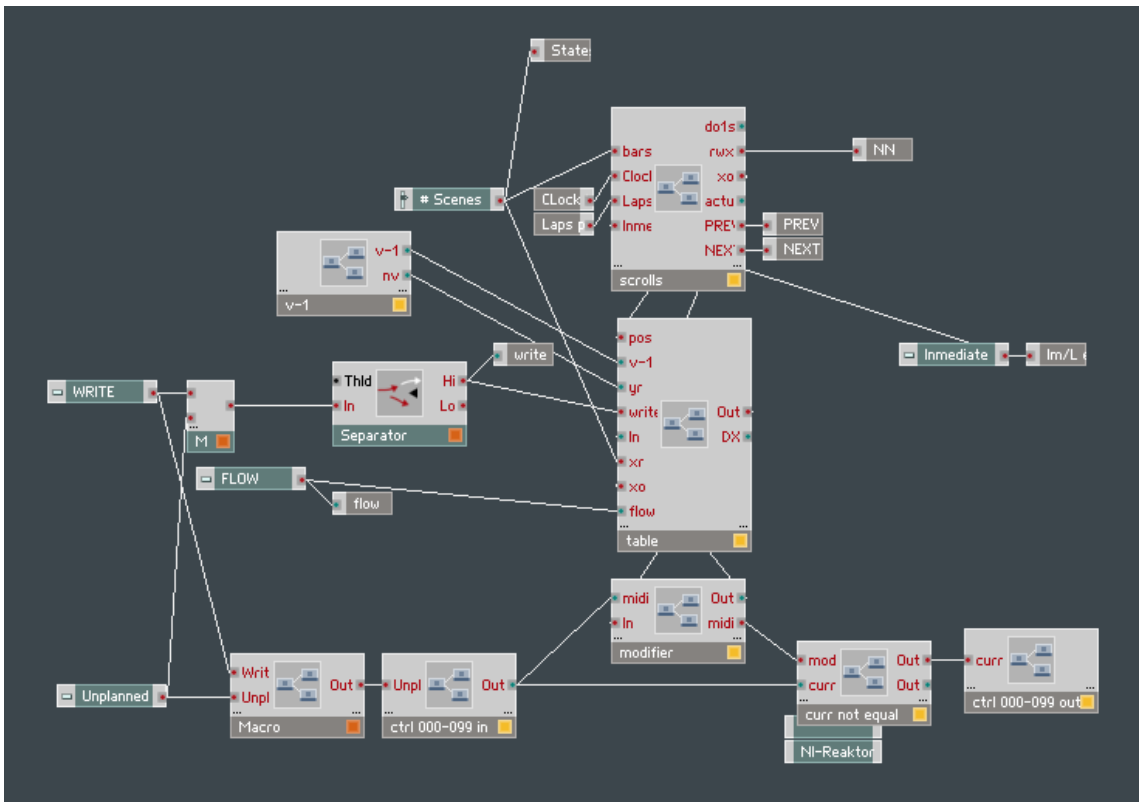



**Figure 12** General blocks of the Scene's memory

2. **Clock gate:** Generation of a binary signal that determines when the loop sweepers start and stop. Rising and falling edges are used to save and recall flags that determine if the state is occupied (with audio) and also its length. Loop sweepers will be explained in fallowing category. Figure 13 shows the logic of this component.

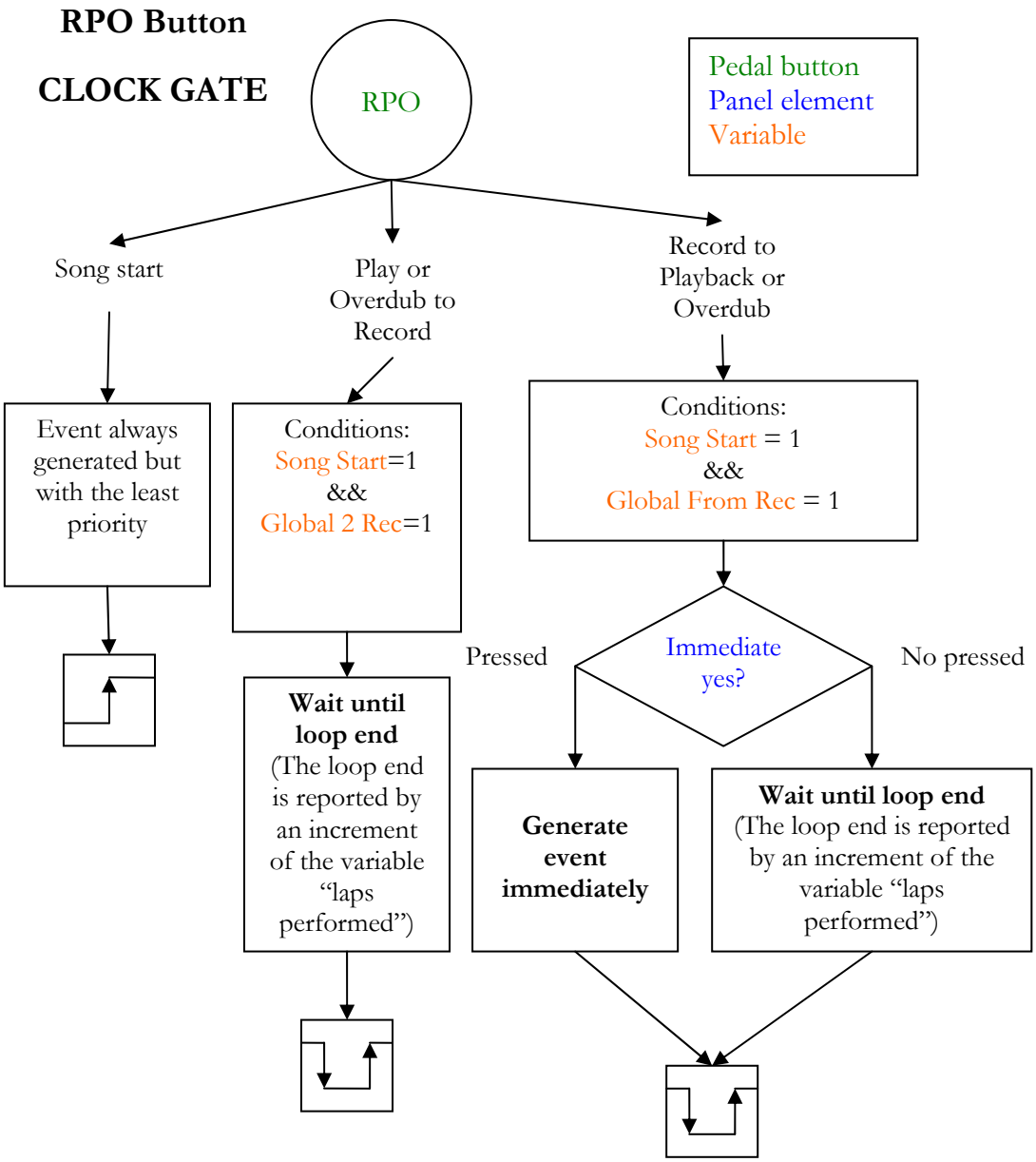

RPO Button
CLOCK GATE
RPO
Pedal button
Panel element
Variable
Song start
Play or Overdub to Record
Record to Playback or Overdub
Event always generated but with the least priority
Conditions: Song Start=1 && Global 2 Rec=1
Conditions: Song Start = 1 && Global From Rec = 1
Pressed
Immediate yes?
No pressed
Wait until loop end (The loop end is reported by an increment of the variable "laps performed")
Generate event immediately
Wait until loop end (The loop end is reported by an increment of the variable "laps performed")

**Figure 13 Effect of RPO button over Clock gate macro.** The clock gate signal is used to start (rising edge) and stop (falling edge) the loop sweepers. The loop sweepers are in charge of pointing consecutives samples at a frequency equal to the sample rate in order to read and write the audio files. The rising edge is used to read flags and arrays that determine whether the state is occupied (with audio) and what's its length. The falling edge is used to store the scene's length when the system is leaving a Record or transfer mode.

The situations where the system has to generate changes in the clock gate are:

- **Song start.**
- **Play or Overdub to Record.**
- **Record to Playback or Overdub.**
- **Change state by pressing PREV/NEXT.**

**Block: Clock gate**

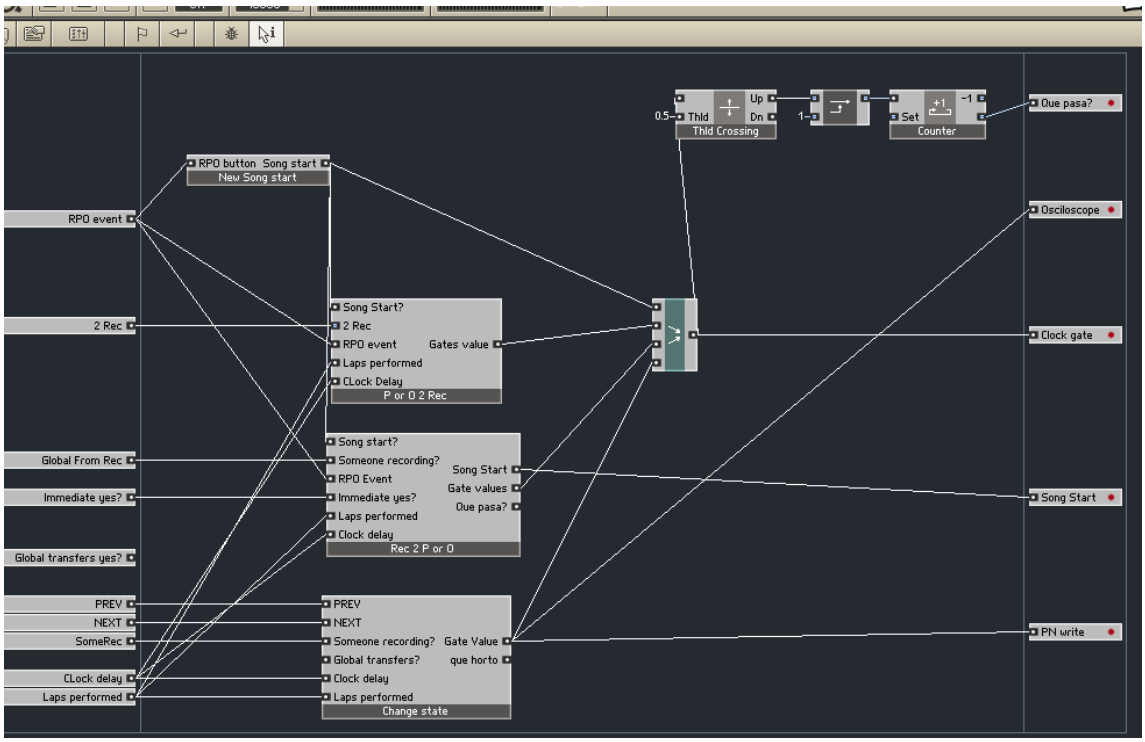



**Figure 14** General Structure of the Clock gate macro**.**

3. **Internal Status:** This macro is in charge of developing the logic to let the user choose the evolution of modes that will occur within a scene. It is also in charge of taking all the decisions, during the performance, regarding the status of each channel and how they interact. The actual status of every scene is based on the RPO evolution list in the panel and other variables like the RPO counter, laps performed, etc (Figure 16).
   - **RPO evolution:** There are five list menus that allow the user to select five modes of operation within a Scene. These lists are placed in chronological order since each mode will occur one after the other and the RPO button in the pedal controller determines the transitions among them. These modes are:
     **-Record**
     **-Playback**
     **-Overdub**
     **-Transfer**
     **-Do nothing**
   - **RPO counter:** counts how many times the user has pressed the RPO button in order to determine the position in the RPO evolution lists. For example if the five RPO evolution lists is Play, Rec, Overdub, Play, depending on the value of this counter the system can be one of these modes.
   - **Channel Status logic:** For each track is determined the actual state. These changes also consider the immediate/loop end configuration.
   - **Global Variables:** Variables used for the clock gate (Global from Rec, Global to Rec, Someone recording?) and flag variables (Someone Play2Rec, used to avoid determining loop length when entering Rec from Play in the same scene)
   - **Block: Internal Status**

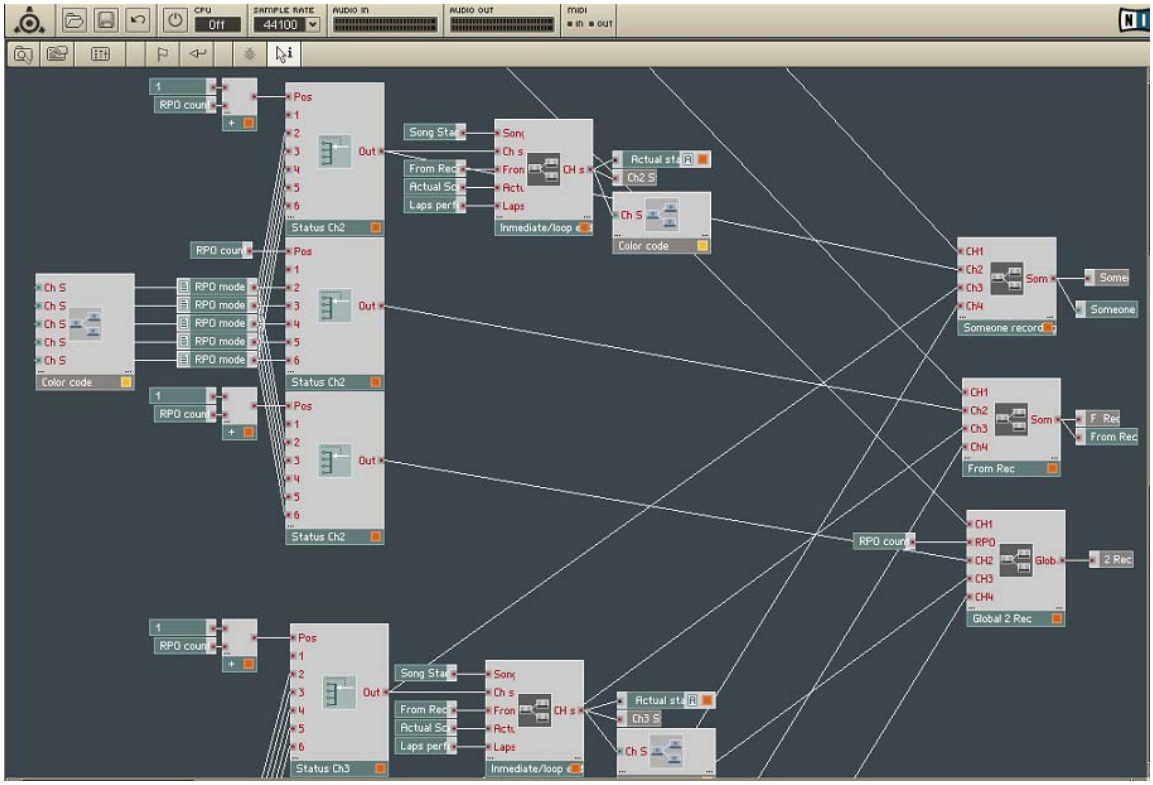



**Figure 15** Part of the general blocks of the Internal Status Macro

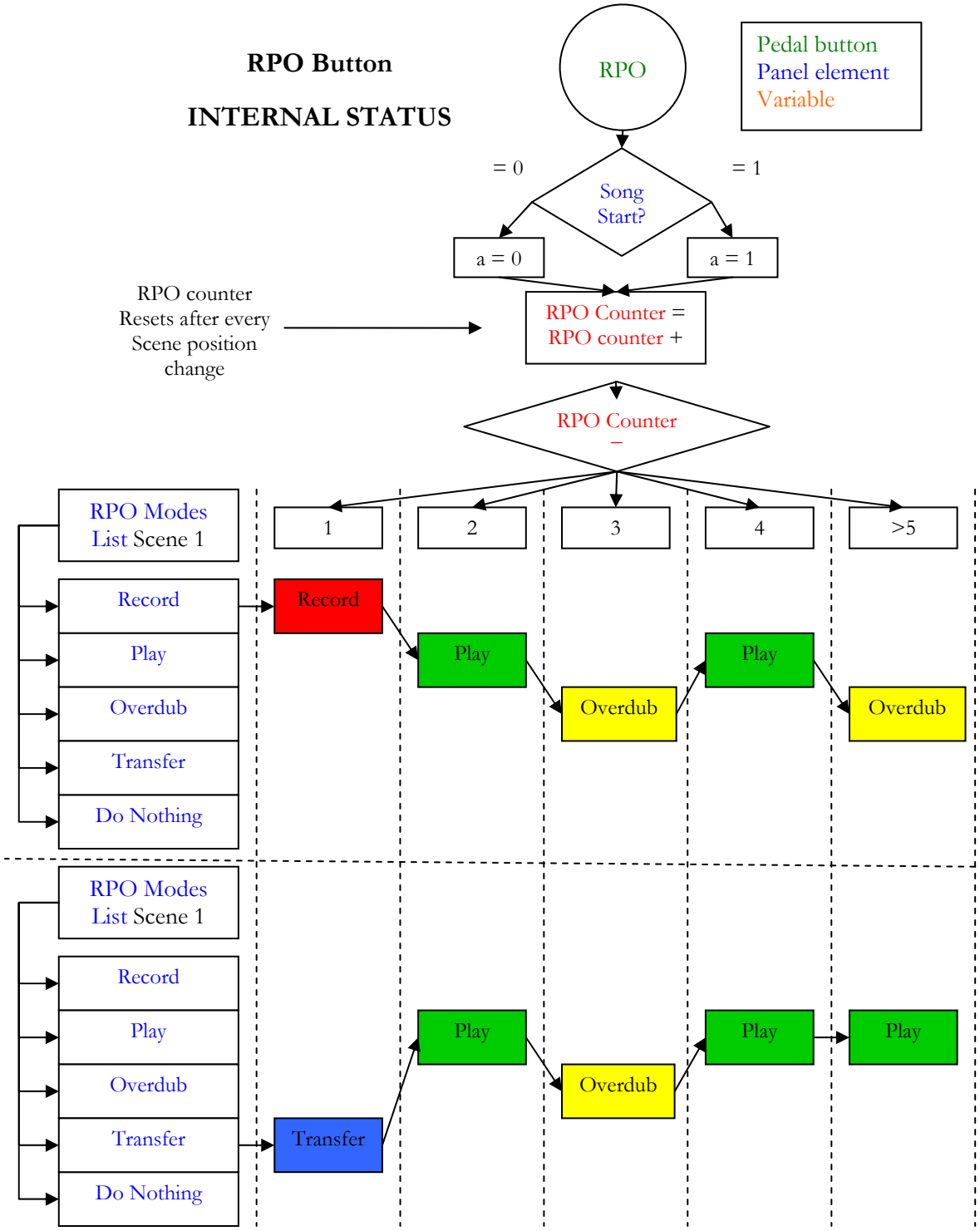
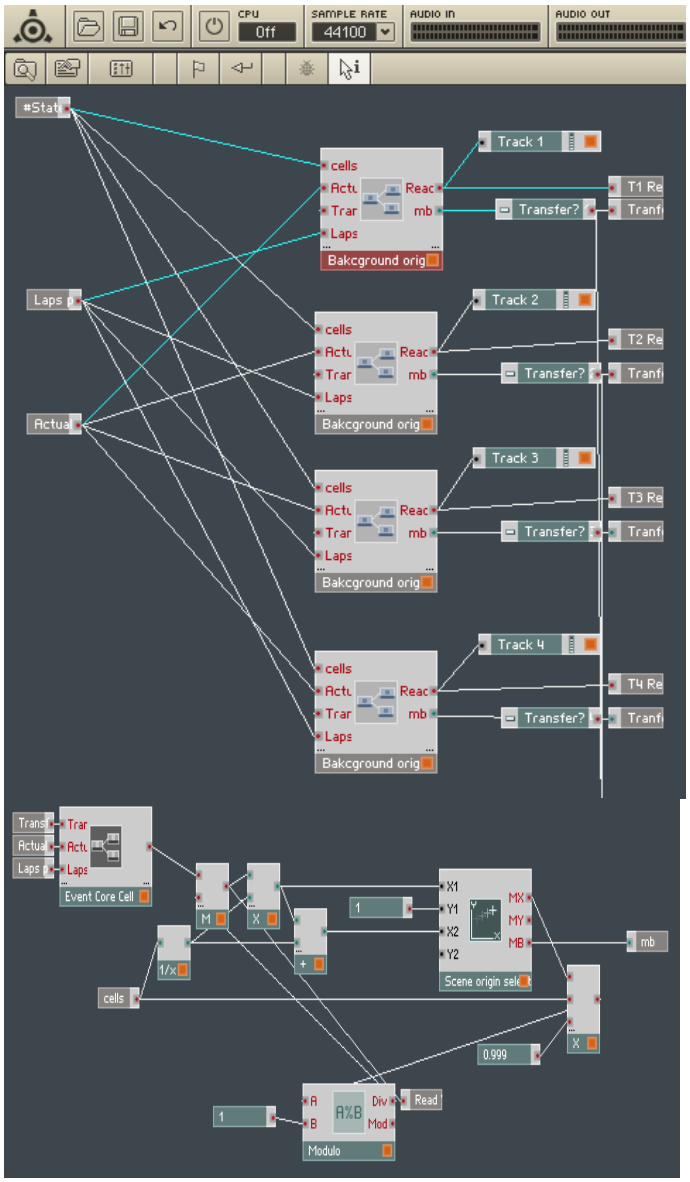



**Figure 16 Effect of "RPO button" over a scene's internal status.** For each track, the user can define 5 transitions in the internal status for each scene. By pressing the RPO button the performer generates the transitions to add all the layers that constitute the scene.

## 4. Next State initialization

This macro manages the audio transfers between scenes. The user has several scroll bars in the panels to select the origin of audio clips that will be imported to the actual Scene

- **Transfer yes? Button:** This button determines if audio will be transferred to that track from a previous scene.
- **Scene origin scroll bar:** Scroll bar that shows all the scenes and where the user will determine the scene where the audio is going to be imported from.
- **Block: Next state Init.**




*General blocks of the macro "Next Scene Init". This is the structure that allows the user to select the audio transfers for the next Scene.*

**Figure 17** Next Scene Init macro

5. **Sample pointer system:** This part of the system is in charge of the logic required to address and manipulate the audio samples stored in each track (Figure 18).

## Scene's length and flag arrays

This part of the system holds the description of each scene regarding if it's occupied or not, and also the length of the scene. Contains also the logic to determine a scene's length based on the length of the other scenes that constitute the background audio.

- **Flags array:** array that contains the flags that indicate if a state is occupied or not. This is used in the read loop sweeper to determine when the pointer has to go back to the initial sample to create the loop.
- **Length's array:** array that contains the length of each state. Every time a new loop is recorded, its length is saved in this array. If the scene had imported audio, then the length is determined by the greater length of the scenes that are in the background.
- **Read and write logic for arrays:** write/read synchronization is crucial in this part of the system
- **Block: Seconds→ Flags & time.**

## Read and write loop sweepers

To store and read audio in the "Audio tables" that represent each track, there are two pointers, called "Read loop sweeper" and "Write loop sweeper". The main variables that determine the functioning of this system are the "clock gate", the values of the "Flag & Time arrays", the "Global transfers", etc. The main components are:

- **Clock gate signal to oscillator:** Depending on the value of the clock gate, it gates the oscillator that is used to feed the counters that implement the sample pointer of both Read loop sweeper and Write loop sweeper.
- **Counter Write position:** moves along the audio table to store samples when recording. When a falling edge occurs in the "clock gate" this block outputs the scene's length that is stored in the "Flag & time block" in order to characterize that state.
- **Counter read position:** Depending on the length read from the "Flag & Time arrays" the counter counts until the length of the scene in order to loop when it reaches this value. It generates a very important output which is the **laps performed counter**. This counter is used in all the situations of the system where it has to wait until a loop ends, for synchronization matters.
- **Approved scene's length:** That a scene has being approved means that the next time is used it wont have the normal RPO evolution, instead it will work only to reproduce the audio that was stored in that scene, either by playing back or doing nothing.

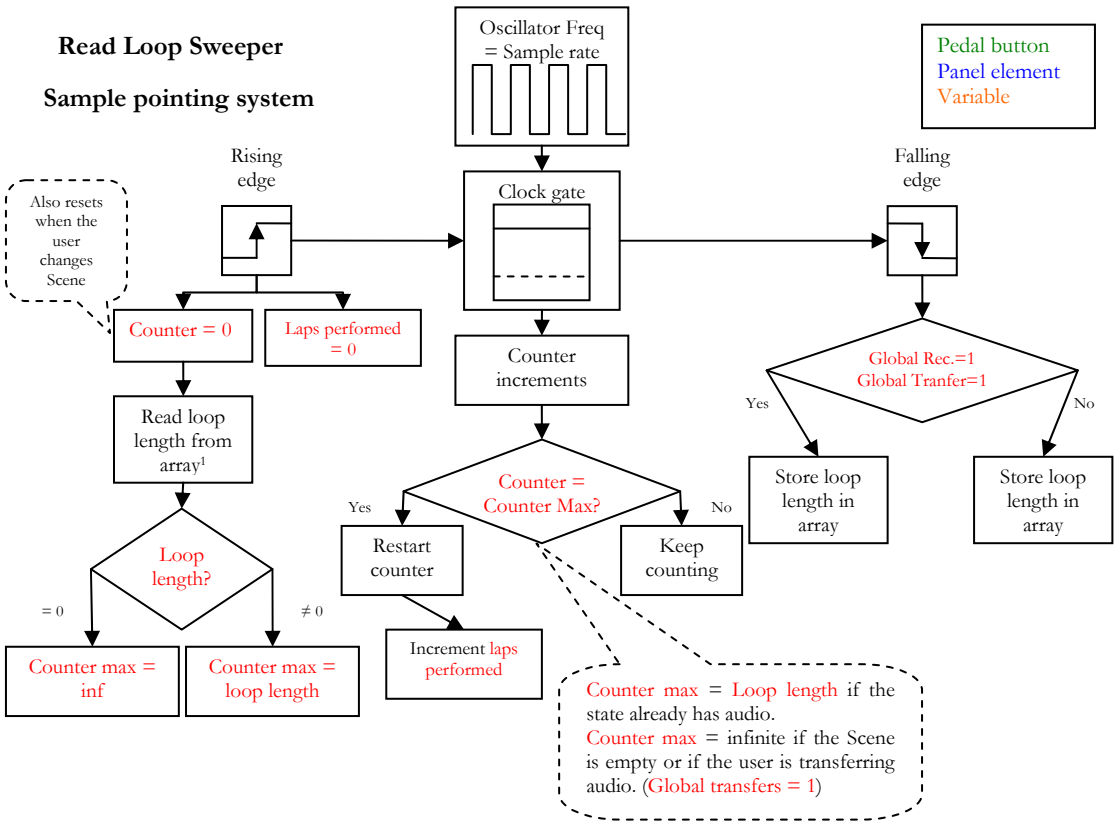
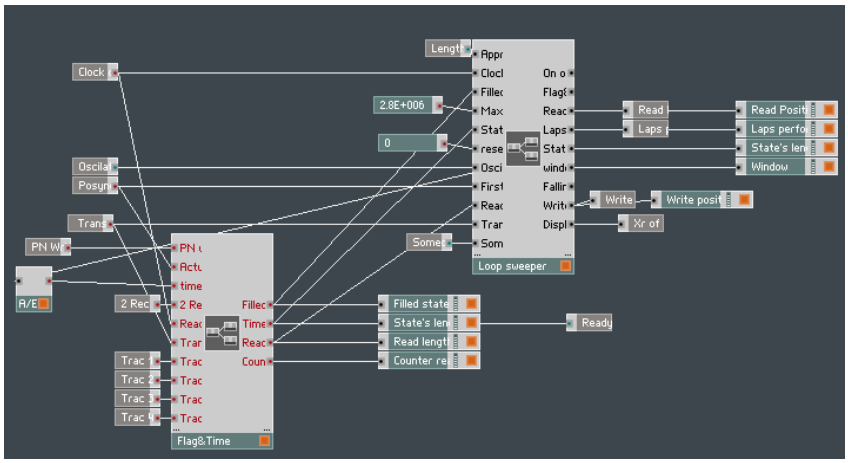
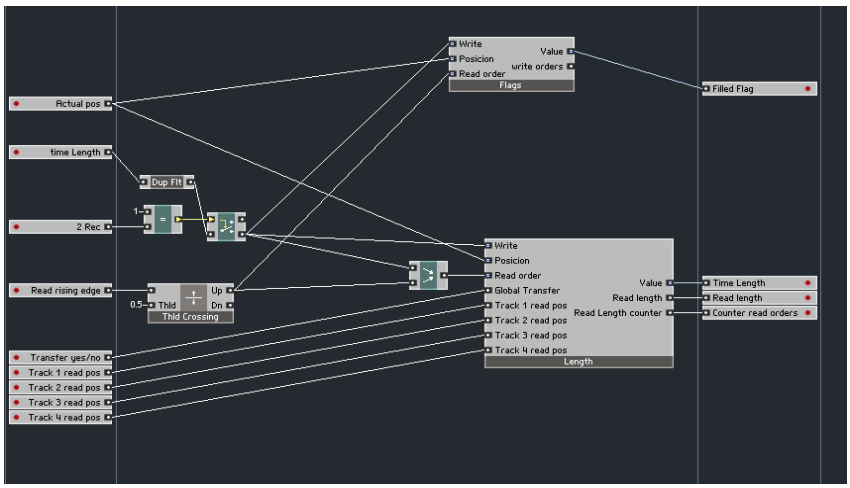
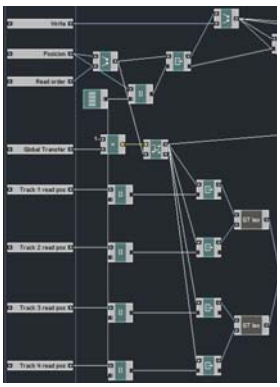



**Figure 18 Simplified read loop sweeper algorithm.** This diagram shows how the clock gate signal allows the oscillator to generate events for the counters. In this "Loop sweeper" part of the program there exist two arrays to stored whether a state is occupied or not, and what's its length (in number of samples). Since the user can be transferring data from different scenes, in [1] the program has to read the length of the scenes that are being transferred and also the actual scene to determine the longest one to use it as the max length of the counter. The variable **Laps performed,** changes every time the read loop sweeper counter reaches the scene's length. This is used as an event to determine when the scene ends. This part of the system could be improved by using a "time stretch" algorithm to make all the imported clips have the same length. This part of the design was out of the projects goals but the actual implementation works well because all clips are referenced to a previous one in a kind of chain, so usually the lengths are equal or an entire multiple.

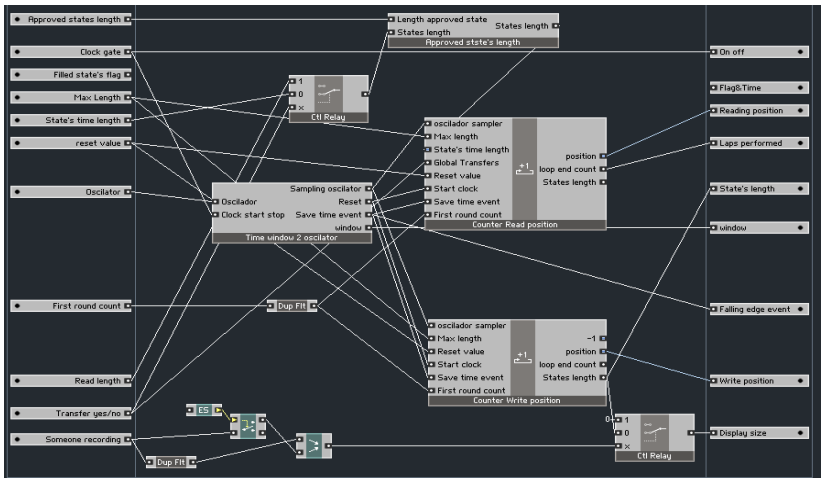


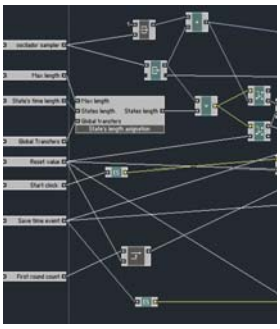

*Sample pointer general blocks. One can see the Flag & Time macro, which contains the arrays that store the lengths of the Scenes while the program is running, and the Loop sweepers that have the counters for reading and writing in the audio table.*



*Internal Structure of the system that is in charge of storing and showing the length of a Scene.*



*Internal Structure of the read Loop sweeper.*

## Audio tracks

This part of the structure contains the audio tables (2 if working with stereo) that will store the audio for each track. There is also a logic needed to control the audio table's inputs and outputs. This system is build through the fallowing blocks and it's also shown in figure19 and 20:

- **Write position:** block that selects if the Read or Write counters will determine the read position. Only in Record and Transfer modes, the position will be determined by the write counter.
- **Input:** Block that determines when the input of the audio table will contain only the line input or the line input plus the previous content of the audio table (overdub). It also has a fader that determines the feedback gain.
- **Read position:** block that determines which state will be read from the table. This is used for audio transfers or overdub.
- **Display content:** the table is addressed in order to show the content of the actual scene. It changes dynamically when the scenes length changes.

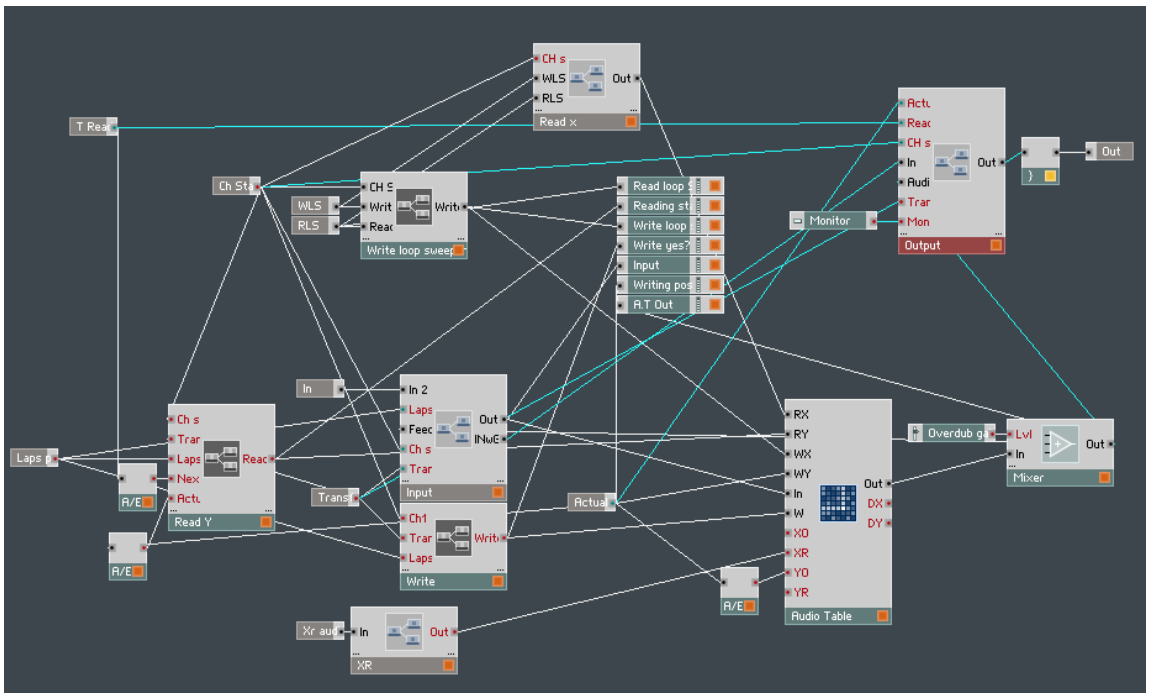



**Figure 19 Audio Tracks general Structure.** Here are the blocks that constitute the structure of the tracks. As one can see there are several macros in charge of taking decisions depending on several control variables. These decisions are made to control the audio table according to the status of the system. For explanations of what this macro does, see figure 20.

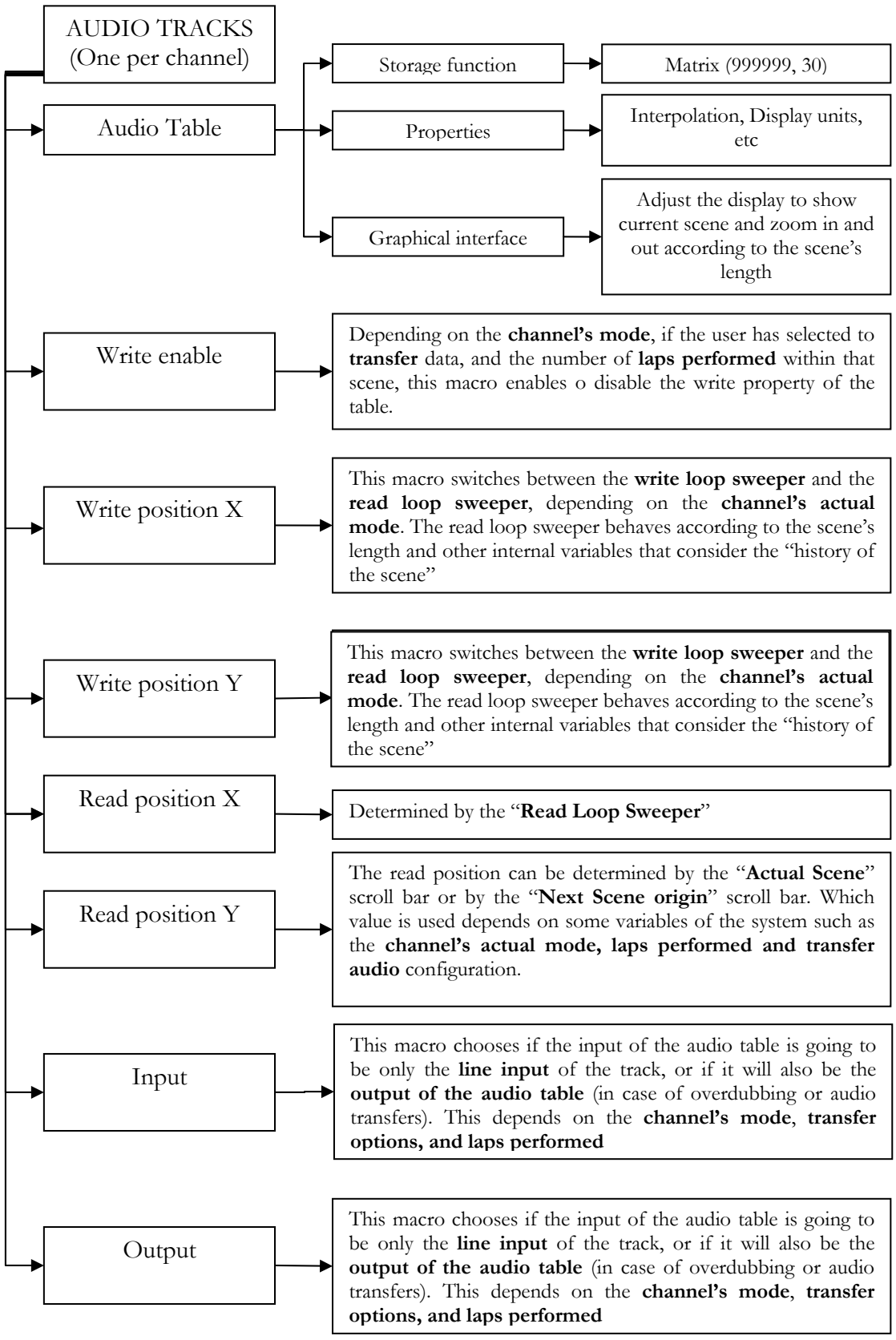

AUDIO TRACKS
(One per channel)
Audio Table
Storage function
Matrix (999999, 30)
Properties
Interpolation, Display units, etc
Graphical interface
Adjust the display to show current scene and zoom in and out according to the scene's length
Write enable
Depending on the **channel's mode**, if the user has selected to **transfer** data, and the number of **laps performed** within that scene, this macro enables o disable the write property of the table.
Write position X
This macro switches between the **write loop sweeper** and the **read loop sweeper**, depending on the **channel's actual mode**. The read loop sweeper behaves according to the scene's length and other internal variables that consider the "history of the scene"
Write position Y
This macro switches between the **write loop sweeper** and the **read loop sweeper**, depending on the **channel's actual mode**. The read loop sweeper behaves according to the scene's length and other internal variables that consider the "history of the scene"
Read position X
Determined by the "**Read Loop Sweeper**"
Read position Y
The read position can be determined by the "**Actual Scene**" scroll bar or by the "**Next Scene origin**" scroll bar. Which value is used depends on some variables of the system such as the **channel's actual mode, laps performed and transfer audio** configuration.
Input
This macro chooses if the input of the audio table is going to be only the **line input** of the track, or if it will also be the **output of the audio table** (in case of overdubbing or audio transfers). This depends on the **channel's mode**, **transfer options, and laps performed**
Output
This macro chooses if the input of the audio table is going to be only the **line input** of the track, or if it will also be the **output of the audio table** (in case of overdubbing or audio transfers). This depends on the **channel's mode**, **transfer options, and laps performed**

**Figure 20** Description of the blocks of the Audio Tracks

# V. Work in progress

## Things remaining to be designed or implemented

1. Be able to **start song** in scene different from zero.
2. **Immediate/loop** end modes to work. For the moment, only the modes of Record and Transfer work with immediate configuration, and Playback, Overdub and Do Nothing work with loop end. This new feature of the program doesn't seem to be really important at the moment even though it will allow the musician to switch Scenes without having to wait until the Scene ends.
3. **Duplicate the maximum length** of the state by multiplying the Read y position of the audio tables for 2 and increment when the loop sweeper reaches 999999. This will allow the musician to have Scenes of 45 seconds for a sample rate of 44100 Hz.
4. **Assignation of all parameters in the MIDI vector**. To be able to program the values of parameters for each scene, the controllers have to be assigned a cell in the MIDI vector. For this the programmer will use the MIDI table shown in the appendix, to decide where to assign each control.
5. Find a way to **approve a state** that does not have to wait until the scene's length is determined. The initial idea is to do this by including an approve procedure when the user presses next. It is assumed that if the user presses NEXT, the Scene was well performed and therefore can be approved.
6. Design a **restart scene system** (with a button from the pedal controller, it could also be the PREV button)
7. **Improve graphical interface** to facilitate lecture of the system. For example use of a color code for the Actual status, and next state status screens.
8. Addition of a **new audio table to allow stereo recordings**. In the actual designed the output is in Stereo Form since the user can pan all the channels. For further improvements of the program, Stereo recording could also be possible by duplicating the "Track" macro. This would be very simple but to avoid CPU load it won't be implemented in this version
9. **Auto record button**. The song and overdub modes could be initiated with the signal coming at the input of the track. In this way the system will be able to synchronize to the musicians gestures without requiring the use of the pedal (for these two cases).
10. **Mute button**. Add one button per track to determine if that track can be muted with a "mute button" on the pedal controller. The user can mute tracks with a gate signal from the pedal controller. This option will give the possibility of making more radical changes in the loops as rhythmic access to layers of a scene.
11. **Momentary punch in**. Design a button to switch between overdub and Rec within a state (momentary punch in). The idea is that only the "input macro" of the audio table is switched from (line in + audio table) to (line in). This could be implemented by a new button in the panel for each track and in the pedal controller, with the same button as the one used in the momentary mute option.
12. **Metronome.** The system could have a metronome to be used as a reference for the musician. The user should be able to select in which Scenes wants to hear the click, and also the tempo.

## Optimization possibilities

Most of these ideas are intended to improve the design. Due to time constraints they were not included in this version of the program.

1. **Group Scenes**. The idea is to be able to create groups of scenes that are played chronologically. This will allow the user to create "Super Scenes" that will be loops of sequences of previous loops.
2. **Create bitmaps and skins to improve the graphical interface**. Some buttons could even be animated to make the interface less static.
3. **Insert new scene between others already programmed**. This will require to be able to transfer a set of scenes to another scenes in order to "open" space for a new scene.
4. **Improve the layer separation in one track**. It will be better to have separate access to all layers that exist in a scene.
5. **Copy audio from different scenes and be able to erase one scene**. In the actual version the user can only the complete audio track, or to erase one scene must record with a zero input in the table. Copying a scene and its audio should also copy the length in the approved states event table.

# VI. Development history

## Performance needs

The first stage consisted in analyzing performance styles in which the musicians combine the use of electronic equipment with their instruments or voice. The goal was to improve the quality of the show by allowing the musician to express their musicality with the assistance of technological equipment, without having the machine as the protagonist of the performance. The performances analyzed were principally the ones of Juana Molina (Argentina), Todos Santos (Venezuela) and Jamie Lidell (Germany).

## Other systems analysis

To understand the different tools used by the artists for their performance, some loop machines and sequencers where analyzed. The most important features were examined and also how these two systems could be combined to create an "enhanced loop machine". The idea was to combine the concept of loop machines with the idea of organizing the different loops along a song like it is done in a normal sequencer in the studio.

For the development of the project, several programming platforms were studied to determine the most suitable option for the project, which was Reaktor from Native Instruments.

## Learning about Reaktor and its possibilities

After doing some tutorials and reading the manuals to understand how to program in this language, several instruments from the Reaktor Library were analyzed to have a better notion of how the final product of the project could work.

Reaktor allows you to open the instruments and see how they were built. This helped to understand more complex structures and also the configurations of the graphical interface.

The initial approach to the project was to use a sequencer and then add the modifications to transform it into the "Scene builder". Unfortunately the sequencers available in Reaktor are for MIDI sequences but not to work as audio editors. In Reaktor the elements that were found useful for the project were the tape decks and audio tables and therefore it became part of the project to design a system capable of storing the audio of the different Scenes.

## No audio sequencers

After finding that there were no sequencers (as the ones use for editing audio Nuendo, Cubase, etc) for recording and playing back the audio, it had to be included in the design, a system to do this.

The substitute of the sequencer had to be a system able to store audio in an organized way, combined with a control logic to explore the different clips of each track that while being graphically intuitive for the user.

## Approaches rejected

**Record in tape decks:** tape decks constitute an important CPU load, and therefore only few of them were allowed to be used at the same time. Some Loop machines designed in Reaktor use two of these elements to be able to Record and overdub, feeding one of them with the output of the other one. This was an efficient solution for overdubbing one loop, but this structure was to basic for the project and did not allow to record different Scenes. To do this several Tape decks would be needed, which would use too many CPU cycles.

**Use the tape deck's "Pos" output to point an audio table:** After deciding that audio tables were more efficient for solving the storage problem, since they had a matrix structure, a system to point the samples along the audio table had to be designed. Initially, the "pos" output of a tape deck was considered for doing this. This option was good in the sense that the control signals that had to be used were the ones for the tape deck (record, loop, stop etc) in order to point the samples in the audio table. The problem was that this system lack flexibility to have a closer grip to the system that would read the samples. There had to be more communication between the pointing system and the memory of the program. Every time a Scene was created, the length had to be determined, stored in an array, as well as other variables that for example show if the Scene was occupied or not.

## Approaches used

The final decision regarding this part of the system was to design from scratch a pointer for the samples and store them in an audio table that would be used as a matrix, in which each Scene is stored as a horizontal vector. Another important advantage of audio tables is that it is possible to read and write at the same time, and the read operation is performed before writing. This is very useful for overdubbing, even though synchronization between Read and Write has to be very well synchronized.

Designing a custom pointing system will allow better interaction with the rest of the logic of the system. This system was implemented by using counters that also informed about number of laps performed, Scene's length as well as different modes for resetting and determining the limits of the counter depending on several variables of the program (Track internal status, laps performed, interrelations with other tracks, etc)

There were design two counters, one for reading and the other for writing. They will work independently and will respond to the actions taken in the pedal controller and configurations of each scene.

## Rec, Overdub, Playback, Transfer and Do nothing modes

To increase the possibilities of the system, the concept of internal status was develop in order to give to the user more options to create every Scene. The concept consists in making possible to evolve how the track will work within each Scene. For this the user

should have the possibility to select the different modes and also the order in which they will be produced.

These transitions should be triggered mainly by the RPO, PREV and NEXT buttons in the pedal controller.

## Approaches rejected

The initial approach was to give the user the option to choose in a menu list, different RPO evolution modes. These evolution modes were for example Rec_Play_Overdub_Play_Overdub and the sequence was determined in a "Fixed Style". For each mode there was a state machine that depending on how many times the user had pressed the RPO button within a Scene, it would change states producing the different modes.

This approach was rejected. Although it works perfectly for one track, when two tracks were used, the modes collided in cases that were impossible for the performer or for the system.

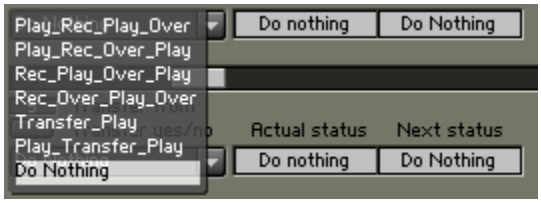



**Figure 21** RPO evolution initial approach

## Approaches used

It was decided to give more flexibility so that "Impossible cases" were avoided. The RPO evolution list was substituted for 5 Internal Modes List. This allows the user to determine 5 internal modes per Scene/Track and therefore be sure that the different tracks will be performing compatible actions.

This system is much more efficient and reduces CPU load that was caused by the logic required for the initial approach.

## Managing scene lengths

One of the most important aspects to manage loops and audio for each scene was how to determine and use the length. This value was not only needed to establish the limit of the counters to loop the samples but also for synchronizing mode changes and automatic tasks that occurred when the loop reaches the end.

To decide when to store the length of a Scene, one must consider the internal status all the tracks in the Scene. The decision on how to use the counters depended in the previous, the actual and the next modes and their relation with users actions with the pedals. Every time the user presses a button of the pedal controller, the system checks what is the status of the system and depending on several options it decides if the length should be stored in the arrays or also if the arrays should be written.

Important variables were created to make these decisions, some of which are mentioned below.

- **Global To Rec**

This variable determines if any of the tracks is going to enter a Record State. This is used for several reasons. For example if a scene starts transferring audio, and after this is going to enter in Record mode, the length of the scene should not be determine until the user closes the Rec mode. Therefore Global to Rec can avoid storing scene's length that is not definitive for the Scene. Another use is that when entering a record mode, the clock gate should to a pulse (101) since this will give the order to the sample pointers to restart the counting and check if the scene was occupied with previous audio in any of the channels.

- **Global From Rec**

This variable determines if any of the tracks is leaving a Record mode. If this is happening, the system should store a new Scene's length in the array and also raise the flag that shows that the state is occupied. This variable also determines that another pulse (101) should be produce in the clock gate signal. This will produce the write order, during the falling edge, and then the read order during the rising edge. This variable is also used to determine if a pedal action (RPO or PREV/NEXT buttons) should be performed immediately or wait until loop end. If the system is leaving a "Global Rec or Transfer Modes" (global mining any of the tracks) the action has to be performed immediately.

- **Global Transfers**

This variable determines if any of the tracks is transferring audio. Audio transfers are not only performed in Transfer modes. A recording mode could also be transferring data, which will work at the end as an overdub of the audio of a previous scene, in the actual one. This variable is used in several macros of the system as for example determining who should be the input of the audio table, if it should be the output of the audio table (feed back) and the line input, or only the line input

- **Someone recording**?

A very important issue in this system is the synchronization of the signals. That is the reason the system checks if the past, actual or future status will be Record or Transfer. In this case the system uses the actual status to determine if any of the tracks is in recording mode. This is use for example to select the inputs of the audio table, the write enable, and which Sample pointer will determine the write position, if it would be the read counter or the write counter.

## Diary of the project and info in the structure of the program

A more detailed explanation of the evolution of the project is available in a Diary of the project. There were many difficulties that had to be solved during the project, but they are not considered so important for this report. Many of them were part of the learning process of using new software and also for being an experimental new approach to the problem. For readers interested in very specific details, the information used to develop the program is registered both in the Diary and also in the properties of each macro that was designed.

# VII. Quality control tests and results

This Section of the report shows the tests that were performed to the system. They are most of the possible combinations that could be made by the user. During this quality control test, most of the problems were solved to secure the functioning of the system.

## Single Track tests

### Internal mode evolution tests: (without transfer button pressed)

| Mode 1 | Mode 2 | Mode 3 | Mode 4 | Mode 5 | Possible Cause | Behavior allowed? |
|---|---|---|---|---|---|---|
| Record | Play | Overdub | Play | Overdub | | |
| **Observations:**<br>Works Perfect.<br><br>Sometimes the signal clips at the beginning | | | | | Fast jump between the last sample of the clip and the first one | yes |
| Record | Overdub | Play | Overdub | Play | Fast jump between the last sample of the clip and the first one | |
| **Observations:**<br>Works Perfect.<br>Sometimes the signal clips at the beginning | | | | | | yes |
| Nothing | Record | Play | Overdub | Play | | |
| **Observations:**<br>Problem<br>The clock gate change from Playback or Overdub to record doesn't occur, I suppose because there is no loop end which is a requisite to produce the clock gate signal change. | | | | | The clock gate is not changing when the mode goes from Do nothing to record | To create a change in the clock signal the scene has to have a length so the P or O to Rec can occur |
| Record | Nothing | Play | Overdub | Play | | |
| **Observations:**<br>Works Perfect. | | | | | | yes |
| Play | Record | Overdub | Play | Overdub | WLS is not restarting when entering a record mode. Laps perform is counting before starting scene | This is probably a forbidden configuration since for play mode there has to be a previous audio and scene length in the array |
| **Observations:**<br>Problem<br>The clock gate change from Playback or Overdub to record doesn't occur, I suppose because there is no loop end which is a requisite to produce the clock gate signal change | | | | | | |
| Record | Play | Record | Overdub | Play | | |
| **Observations:**<br>Works Perfect. | | | | | | yes |
| Record | Nothing | Record | Overdub | Play | | |
| **Observations:**<br>Works Perfect. | | | | | | yes |
| Play | Overdub | Play | Overdub | Play | | |
| **Observations:**<br>Problem<br>There is no length for the Scene, so it Changes state but never restarts the loop sweepers. | | | | | | This is probably a forbidden configuration since for play mode there has to be a previous audio and scene length in the array |
| Overdub | Play | Overdub | Play | Play | | |
| **Observations:**<br>Problem<br>There is no length for the Scene, so it changes state but never restarts the loop sweepers. | | | | | | This is probably a forbidden configuration since for play mode there has to be a previous audio and scene length in the array |
| Record | Play | Overdub | Play | Overdub | | |
| **Observations:**<br>Works Perfect.<br><br>**USING Next button to close loop** | | | | | | yes |

## Internal mode evolution tests:
## (with "transfer" button pressed, but Next Scene Init = Prev Scene)

| Mode 1 | Mode 2 | Mode 3 | Mode 4 | Mode 5 | Possible Cause | Behavior allowed? |
|---|---|---|---|---|---|---|
| Record | Play | Overdub | Play | Overdub | Length of the Scene wrong due to a wrong save event. This event was generated because of a synchronization problem in the variable 2 Rec of the status macro | |
| **Observations:**<br>Problem<br>The display is showing a wrong size.<br><br>Solved<br>**Solution**<br>Place a Latch inside the macro for creating the "2 Rec" variable<br>This ensures synch between Ch mode changes and RPO | | | | | | Yes. |
| Record | Overdub | Play | Overdub | Play | | |
| **Observations:**<br>Works perfect | | | | | | |
| Nothing | Record | Play | Overdub | Play | | In this case it works well since there is a Scene length which is determined by the Prev Scene |
| **Observations:**<br>Works perfect | | | | | | |
| Transfer | Play | Overdub | Play | Overdub | | |
| **Observations:**<br>Works Perfect | | | | | | yes |
| Transfer | Overdub | Play | Overdub | Play | | |
| **Observations:**<br>Works Perfect | | | | | | |
| Transfer | Record | Play | Overdub | Play | | The same action could be performed using a Play_Rec_Play_Over mode, since the first mode will import audio, and after that the user can record |
| **Obervations:**<br>Small Problem<br>Transfer works well. Record works in the first lap<br>as Playback, and after the second lap behaves like Overdub | | | | | | |
| Transfer | Nothing | Overdub | Play | Overdub | | |
| **Obervations:**<br>Works perfect | | | | | | yes |
| Record | Nothing | Play | Overdub | Play | | |
| **Obervations:**<br>Works perfect | | | | | | yes |
| Play | Record | Overdub | Play | Overdub | | |
| **Obervations:**<br>Works perfect | | | | | | yes |
| Record | Play | Record | Overdub | Play | | Probably this mode is not required in the performance.<br>If the user wants to use audio of this scene after playback, there could be a Rec mode in next Scene |
| **Obervations:**<br><br>Small Problem<br>Its reading some information that is not the one stored in the actual Scene. It seems to be the one of the previous scene but with a different length. | | | | | | |
| Record | Nothing | Record | Overdub | Play | | |
| **Obervations:**<br>Small Problem<br>Same as previous test | | | | | | Same as previous test |
| Play | Overdub | Play | Overdub | Play | | |
| **Obervations:**<br><br>Works Perfect | | | | | | yes |
| Overdub | Play | Overdub | Play | Play | | |
| **Obervations:**<br><br>Works Perfect | | | | | | yes |

# Tests for two tracks

## Internal mode evolution tests: (Starting song)

| Track | Mode 1 | Mode 2 | Mode 3 | Mode 4 | Mode 5 | Observations |
|---|---|---|---|---|---|---|
| 1 | Record | Play | Overdub | Play | Overdub | |
| 2 | Nothing | Record | Play | Play | Play | |
| **Test 1**<br>**Observations:**<br>Problem<br>When pressing RPO button of the first track, since the system is entering a new record mode in the same scene, the length of the first clip is not stored and therefore the RLS = RWS | | | | | | This configuration can be performed by using two Scenes. In the first one Track 1 records and in the second one track 1 transfers while two records This is shown in **Test 4** |
| 1 | Record | Overdub | Play | Play | Play | |
| 2 | Nothing | Record | Play | Overdub | Play | |
| **Test 2**<br>**Obervations:**<br>Problem<br>When pressing RPO button of the first track, since the system is entering a new record mode in the same scene, the length of the first clip is not stored and therefore the RLS = RWS | | | | | | This configuration can be performed as It is shown in **Test 5** |
| 1 | Record | Play | Play | Overdub | Play | |
| 2 | Nothing | Record | Play | Overdub | Play | |
| **Test 3**<br>**Obervations:**<br>Problem<br>When pressing RPO button of the first track, since the system is entering a new record mode in the same scene, the length of the first clip is not stored and therefore the RLS = RWS<br><br>Conclusion:<br>Only 1 track "Recording" track per Scene | | | | | | This configuration can be performed as itis shown In **Test 5** |

## Internal mode evolution tests: (Transfer yes first track)

| Track | Mode 1 | Mode 2 | Mode 3 | Mode 1 | Mode 2 | Mode 3 | Mode 4 | Mode 5 |
|---|---|---|---|---|---|---|---|---|
| 1 | Record | Play | Overdub | Transfer | Play | Play | Play | Play |
| 2 | Nothing | Nothing | Nothing | Record | Play | Overdub | Play | Overdub |
| **Test 4**<br>**Obervations:** | Works perfect<br><br>Also Using NEXT button to close first loop | | | | | | | |
| 1 | Record | Play | Play | Record | Play | Overdub | Play | Play |
| 2 | Nothing | Nothing | Nothing | Record | Play | Overdub | Play | Play |
| **Test 5**<br>**Obervations:** | Works perfect<br><br>Also Using NEXT button to close first loop | | | | | | | |
| 1 | Record | Play | Overdub | Transfer | Play | Play | Play | Play |
| 2 | Nothing | Nothing | Nothing | Record | Play | Play | Play | Play |
| **Test 6**<br>**Obervations:** | Works perfect | | | | | | | |
| 1 | Record | Play | Overdub | Record | Play | Play | Overdub | Play |
| 2 | Nothing | Nothing | Nothing | Nothing | Nothing | Record | Play | Play |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **Test 7 Obervations:** | Careful in its use. When using a Second record mode within a Scene, a new length of the Scene is determined. Therefore if the second record mode is longer then the first one, the RLS will be equal to WLS and therefore pre-existing audio in the table will be played back. Alternative, to make second record equal os smaller or Split Scenes. | | | | | | | |
| **1** | **Record** | **Play** | **Overdub** | **Play** | **Play** | **Play** | **Play** | **Play** |
| **2** | **Nothing** | **Nothing** | **Nothing** | **Nothing** | **Record** | **Play** | **Play** | **Play** |
| **Test 8 Obervations:** | Problem During the first 2 modes of this configuration there is no problem. When one changes to the third one if the length of the record mode is longer then the play some previous recorded audio will sound to fill the length. The solution is to configure it as it is shown on **Test 9** | | | | | | | |
| **1** | **Record** | **Play** | **Overdub** | **Play** | **Transfer** | **Play** | **Play** | **Play** |
| **2** | **Nothing** | **Nothing** | **Nothing** | **Nothing** | **Record** | **Play** | **Play** | **Play** |
| **Obervations: Test 9** | Works perfect | | | | | | | |

## Verification of audio inputs and outputs in the internal structure of the software:

| | | | Audio table in | Audio table in | Mixer | Mixer | | | |
|---|---|---|---|---|---|---|---|---|---|
| **Mode** | **Transfers?** | **Next Scene init** | **Feed back** | **Input = Line** | **Output = A.T** | **Output = Line** | **WLS** | **RLS** | **Observations:** |
| **Record** | No | Actual Scene | No | Yes | Yes | Yes | WLS | First sample | If Transfer? Is not pressed, then Actual Scenes is always equal to Next Scene Init |
| **Play** | No | Actual Scene | No | No | Yes | Yes | RLS Actual Scene | RLS Act Scene | |
| **Overdub** | No | Actual Scene | Yes | Yes | Yes | Yes | RLS Actual Scene | RLS Act Scene | |
| **Transfer** | No | Actual Scene | Yes | No | Yes | Yes | RLS Actual Scene | RLS Act Scene | |
| **Do Nothing** | No | Actual Scene | No | No | No | No | X | RLS Act Scene | |
| **Record** | Yes | Prev Scene | Yes | Yes | Yes | Yes | WLS | RLS Prev Scene | |
| **Play** | Yes | Prev Scene | No | No | Yes | Yes | RLS Prev Scene | RLS Prev Scene Lap 1, | After that, RLS Actual Scene |
| **Overdub** | Yes | Prev Scene | Yes | Yes | Yes | Yes | RLS Prev Scene | RLS Prev Scene Lap 1, after | After that, RLS Actual Scene |
| **Transfer** | Yes | Prev Scene | Yes | No | Yes | Yes | WLS | RLS Prev Scene Lap 1, after | After that, RLS Actual Scene |
| **Do Nothing** | Yes | Prev Scene | No | No | No | Yes | XXXXXX XXXXXX XX | RLS Prev State | Since the line in is connected to the output one can play with this configuration |

These tests were performed to assure that all the variables related with the audio inputs and outputs were behaving correctly during program debug

# VIII. Appendix

## Terminology

*Template*
A form that contains all the configurations of the program for each scene. Represents the result of the programming stage of the software and it is used as the framework for the performance.

*Scenes*
Scenes are time intervals where the program has a fixed configuration and the musician is performing with the same instruments.

*Clips*
A clip is the audio contained in one track of one scene. All the *clips* in a scene represent all the audio contained in that scene.

*Overdub*
*Overdub* is to record over a previous existing file. The audio is superimposed, so thereafter the resulting audio file can not be separated into its components.

*Scroll bar*
An element in the front panel that allows the user to select a scene from the song.

*Clients*
When two or more elements of the program are accessing data from the same table, they are called *clients*.

*Ensemble*
A combination of instruments build in Reaktor is called an *ensemble*. Scene builder is an *ensemble*.

*RPO evolution*
Within each scene, a track can be performing in different modes: recording; overdubbing, playing, transferring audio or doing nothing. The *RPO evolution* is a sequence of these internal modes of the Scene. You can select one of the modes from a list, and then by pressing the RPO button in the pedal you will switch between them in order to create the clip for your scene.

*Macro*
Macro is a generic name for structures or building blocks made in Reaktor. Most of the *macros* of Scene builder are made from using the core structure (lower level of this programming language)

# Scene Builder

*by Leonardo Aldrey*

User Manual

TABLE OF CONTENTS

1 What is Scene Builder? ..... 4
2 System Requirements and Recommendations ..... 5
2.1 Hardware ..... 5
2.2 Software ..... 5
3 About the manual ..... 5
4 Hardware Connections ..... 5
4.1 Audio Channels ..... 5
4.2 Foot controller ..... 5
5 Elements of the front panel ..... 6
5.1 Output mixer ..... 6
5.2 Scene navigator ..... 6
5.3 Tracks ..... 8
6 Quick Start Guide ..... 9
7 Programming ..... 10
7.1 Audio Settings ..... 10
7.2 ..... 10
7.3 Next State initialization ..... 10
8 Performance ..... 12
8.1 Using the RPO button ..... 12
8.2 Use of approve and disapprove buttons ..... 12
9 Pre-configuration (Event table) ..... 13
10 Adding new effects to the audio tracks ..... 14
11 Appendix ..... 15
11.1 Technical information about the program structure ..... 15
11.2 MIDI assignations for the scene memory ..... 15
11.3 Terminology ..... 18

# Operation Manual

## 1 What is Scene Builder?

**Scene Builder** is a computer program that allows you to create multitrack recordings during live performance. After programming your song, you will be able to perform with several instruments and record the different layers of your composition by using only a foot controller. It is designed for multi-instrumentalist musicians that intend to create their background tracks live. By programming your song, you generate a *template* in which all your manipulations of the software are stored. During the performance you have only to perform and fill the *template* by using the foot controller.

**Figure 1 Partial View of Scene Builder.** This is a partial view of the graphical interface of scene builder. Here one can see three tracks with some of the parameters thar have to be configured for each scene. Among ths parameters are: internal modes evolution (play, record, overdub, transfer, etc) input and feedback gains, monitor input, origin of audio transfers, etc. On the upper part is the Actual Scene scroll bar which shows in what stage of the song the user is performing. There is also a display to show the audio data of that particlar scene, for each track.

# 2 System Requirements and Recommendations

The requirements are the same as those for REAKTOR 5. A computer with the following minimum specifications is needed:

## 2.1 Hardware

- Pentium III 1 GHz/ Athlon XP 1.33 GHz.
- 512 MB RAM
- Up to 1GB free HD space.
- A sound card compatible with Windows XP
- CD drive

## 2.2 Software

- Windows XP
- Reaktor 5 or later

In addition Scene Builder requires a **MIDI foot controller** with at least 6 pedals and a **MIDI interface**. The described setup is for a Behringer FC1010 foot controller, but the modifications needed for a different controller are also included in the manual.

# 3 About the manual

The **main instructions** to configure every instrument in the Scene Builder's front panel of Reaktor, are found in the **B-View**

Terms that are explained in the **terminology** section are written in ***italics*** font.

In the front panel of the program you can **access the properties of any element** by right clicking it and then selecting "Properties". This will allow the adjustment of its behavior, default values, range, graphical properties, etc.

# 4 Hardware Connections

## 4.1 Audio Channels

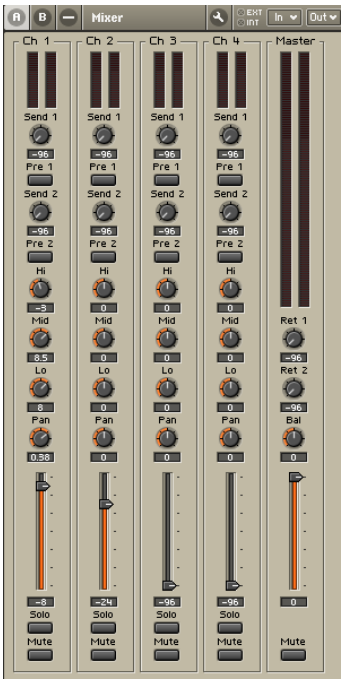
Scene Builder allows the use of four tracks. Connect your instruments or microphones to the audio card. The input for each track is already defined in the program, being Track 1 the correspondent to channel 1.

## 4.2 Foot controller

If your foot controller only has a MIDI output, you will need to use a MIDI interface. Connect the foot controller (or MIDI interface) to the computer.

# 5 Elements of the front panel

## 5.1 Output mixer




- Equalization
- Panning
- Level
- Sends
- Solo/Mute

This Mixer modifies the signal coming from the audio table into the outputs of the audio card. Its controllers are stored in the event table allowing further adjustments like equalize, pan, control level and other parameters in a different way for every scene.

**Figure 2** Mixer for the four tracks

## 5.2 Scene navigator

**# Scenes fader**

This fader determines the number of scenes that the song will have. As a default, the maximum number of scenes is set to 30, but this can be modified in the “x” size of the table.

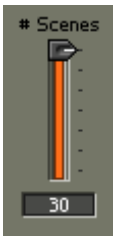



**Figure 3 Number of Scenes fader**

### Current Scene scroll bar

This is a *scroll bar* that lets you navigate the different scenes of the song. The position of the object changes when you press the "NEXT" and "PREV" buttons in the pedal controller. It has also a numeric readout to show the number of the Scene.

**Figure 4 Current scene navigator**

**Planning buttons** immediate/loop end, write, unplanned, flow, approve, and disapprove.

The first four planning buttons are controlled in the front panel of the software. These buttons are used during the programming stage of the song. The last two buttons are included in the pedal, since approving or disapproving scenes would be an action taken during the performance.

- Immediate: Selects if the scene is changed immediately or if it waits until loop end.
- Write: Stores the values of all the parameters of the panel in the event table. The position in the event table is determined by the "Actual Scene" *scroll bar*. The "Event table" is colored in orange when the user writes in this cell of the event table.
- Unplanned: Tells the system that the current state is not planned. The cell in the event cell will turn black to inform the user that the state has not being programmed yet.
- Flow: when this button is selected, a change of the actual scene will modify the values of all the parameters in the program, to the ones that correspond to that Scene. If it is not selected, you can change Scenes while keeping unchanged the parameters of the system.
- Approve: If a scene has been performed correctly, you can approve it with this button. This will allow you to recall the scene, only for playback. When a Scene has been approved, the corresponding cell in the "Approve Scenes Table" will turn orange.
- Disapprove: To inform the system that the scene has to be performed or that if the user recalls it, it will be restarted as it was planned for the first performance.

**Next Scene init**: Audio origin *scroll bar*, transfer yes/no button, *RPO evolution*

"Next Scene init" is a segment of the "Scene Navigator" that shows how each scene will be initialized. This initialization is done for all four channels in order to determine which will be their role in each scene.

- Audio Origin *scroll bar*: This *scroll bar* lets you select from which previous scene you would like to import audio. To confirm your selection of audio to be transferred use the "transfer yes/no" buttons.
- Transfer yes/no button: Select this button when you want a particular channel to import audio from previous scene.
- *RPO evolution:* For each track there are five lists to select the *RPO evolution* of the actual Scene. This modes will be used from left to right and you switch them by using the "RPO button in the foot controller"
- Current status readout: It is a Screen that shows what is the actual mode or status of the scene e.g. recording, playback, *overdub*, etc.
- Next status readout: It is the second screen that presents the initial mode or status of the *next* scene. This gives you the chance to prepare for entering the next state. A color code is used for a easier lecture of this values.

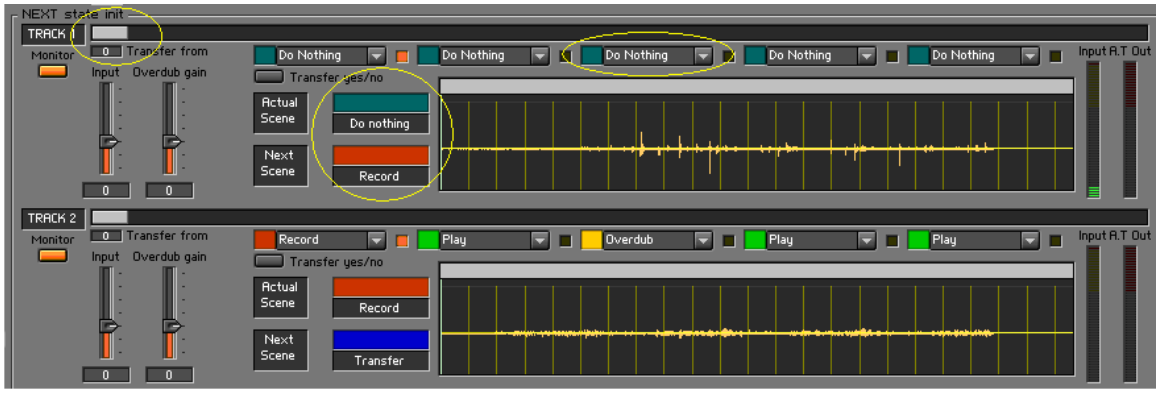



**Figure 5** RPO evolution lists, audio transfer's origin and Color coded screens to report actual and next status.

## 5.3 Tracks

This is where all the audio signals are stored during the performance. In the front panel the main elements are a graph that shows the audio signal, and two faders to select the Feedback gain, and the gain at the input.

**Figure 6** Selecting input gain and feedback gain

# 6 Quick Start Guide

1. Duplicate the original folder of “Scene Builder” that contains the *ensemble* and the table.
2. Change the name of the *ensemble* for the name of the song.
3. Open the *ensemble* with Reaktor 5 or later versions.
4. Connect your foot controller to the MIDI interface and also your instruments to the audio card.
5. Connect the output of your audio card to your speaker system or headphones.
6. Fallow the instructions presented in “Programming stage” section of this manual to create the *template* of your song.
7. Practice your song and adjust the programming.
8. Perform with Scene builder and enjoy.
9. For other songs, repeat all the steps mentioned above.

# 7 Programming

The programming stage is when you will **create the *template* of your song**. Here you will configure the role of all the elements of the system for each Scene. For this you will have to compose your songs always thinking of what you are capable of performing, and how you will perform and distribute the different musical elements that constitute your song.

## 7.1 Audio Settings

1. **Connect all the instruments** that you will be using.

2. Check on the "Audio table input" display if the signal has an **appropriate input level**. If not, adjust the input gain. You can also adjust this in your audio card. Default value for the input gain will be 1.

3. **Adjust** in the **Mixer** the equalization of each channel and also adjust the level for an appropriate interaction between them.

4. For the Levels of the tracks in the mixer, these values can be programmed differently for each scene in order to be able to modify the dynamics of the song. Because of this, you will have to **set the default value of the level faders**. This is done by accessing the properties of the fader and writing in the "default" cell, the value shown by the fader's readout.

## 7.2 Next State initialization

Here you will define what is going to happen in every track on the next Scene. This includes the role of every track in the scene, the *RPO evolution*, audio transfers, etc.

5. Choose the *RPO evolution* for each track. You can choose from five lists, a sequence of modes in which you can use the track within a scene. The first mode in the list will be the one used when you enter the scene. Afterwards if you press the "RPO button", the channel mode will change to the one indicated by the next list. The list and some options are shown in the figure 7

**Figure 7** RPO evolution lists

When selecting these *RPO evolution* modes for all tracks, you have to consider that when the RPO button is pressed, all tracks will change their mode, hence the modes have to be thought of as evolving together.

**Example:** In the first your starting mode has to be record. The first time you press RPO the song will start and then when you press it again the channel will change to Playback and you will hear the last audio clip that you performed. If you press the RPO button, this will allow you to *overdub*, which means that you will be recording over the last *clip* in that *scene.* This internal evolution will allow to ad different layers within a scene.

6. **Select the audio data transfers**. In the scenes where you want to import an audio clip from a previous scene, you will have to choose from which state you want to transfer data, and also activate the "transfer yes" button in order to confirm that audio will be imported. This is done in the following elements of the front panel.

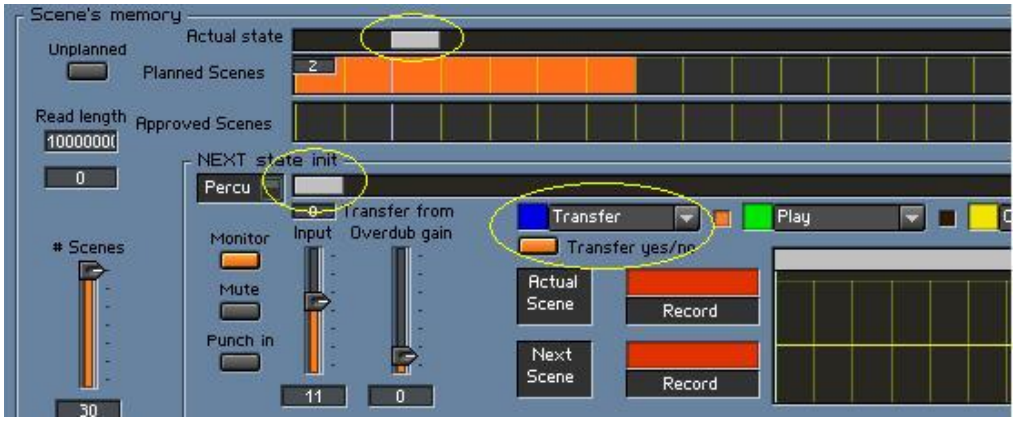



**Figure 8** Configuring audio transfers for one track

In Fig 8 one can see that for track 1, audio is being transferred from the scene 1 into the scene 3 (the first scene is called scene 0). For each track there is a "Next state init" *scroll bar*. This bar is where you will select the origin of the clip that you will be transferring. The "Transfer yes/no" button turns orange when it is selected.

7. **Configuring effects**. If you are using effects for any channel, configure them so they meet your needs for the scene.

8. **Store scene settings**. After you have done all the steps mentioned above, press the "Write" button in order to store these settings into Scene Builder. Repeat steps 5 to seven for all tracks and scenes of your song. After you have done this, you are ready to perform your song, by navigating the "*template*" you have created by programming your scenes.

9. **Adjusting the *template***. If you are not satisfied with the configuration of a scene, you may reconfigure it. To do this, press the "Unplanned" button first. This will make black the cell of the "Event table" in order to inform the user that the scene has to be configured. Repeat steps from 5 to 8 until you are satisfied with the configuration.

# 8 Performance

You have now created a particular "*template*" that only needs to be filled with the audio signal you produce during your performance. All the different *clips* that you perform will assemble like a puzzle. The smaller pieces are the *clips* you perform. Several *clips* from different tracks, played at the same time, will be your scenes, and all the scenes together will build your song.

## 8.1 Using the RPO button

The RPO button in the foot controller determines the **transition between the internal modes** of a Scene. Depending on the actual mode of a scene, the transition can be performed immediately (Living record or transfer modes) or it could wait until the loop end (Playback or *overdub*).

To know what the actual mode is, the panel has a readout called "Actual Status" where you can see the mode that you are performing in for each channel. In addition there is another readout called "Next Status" that informs what will be the first mode of the next scene. This allows you to know where you are standing and also where you are going to. These readouts are shown bellow.

**Figure 9** Reporting the status of the Scene. Color code for better recognition

## 8.2 Use of approve and disapprove buttons

In order to **save a scene during a performance**, you have to use the "Approve/Disapprove" buttons. This works as a stamp that you will place in the scene in order to inform the system that the Scene has being performed correctly and therefore it could be recalled later to use it as a background track. There is another way of recalling a scene without having to approve it which will be to recall it by performing an audio transfer of all the *clips* and tracks of the original state.

In this version of the program there will not be use this approve and disapprove buttons. Scene recalling will be limited to transferring the hall scene.

# 9 Pre-configuration (Event table)

This procedure is not needed if the user duplicates the original folder of the Scene Navigator. This folder already contains the table that you need to run the program.

For each song created in Scene Builder, there has to be a folder in the computers hard drive, where the ***ensemble*** and **event table** are stored.

The **Event table** is used to store the value of all controllers of the front panel for each scene. This table has to have a size of x = 30, y = 100. The "x" component represents the maximum number of scenes and "y" represents the number of parameters that can be stored (limited by the number of voices in the instrument). This table is shared by two "*clients*" or elements in the program.

To create this table, right click on the "Event table" and ten left click on "Properties". Once you have the Properties window, configure it as its shown bellow and then click on the image to store it in the same folder as the *ensemble*.

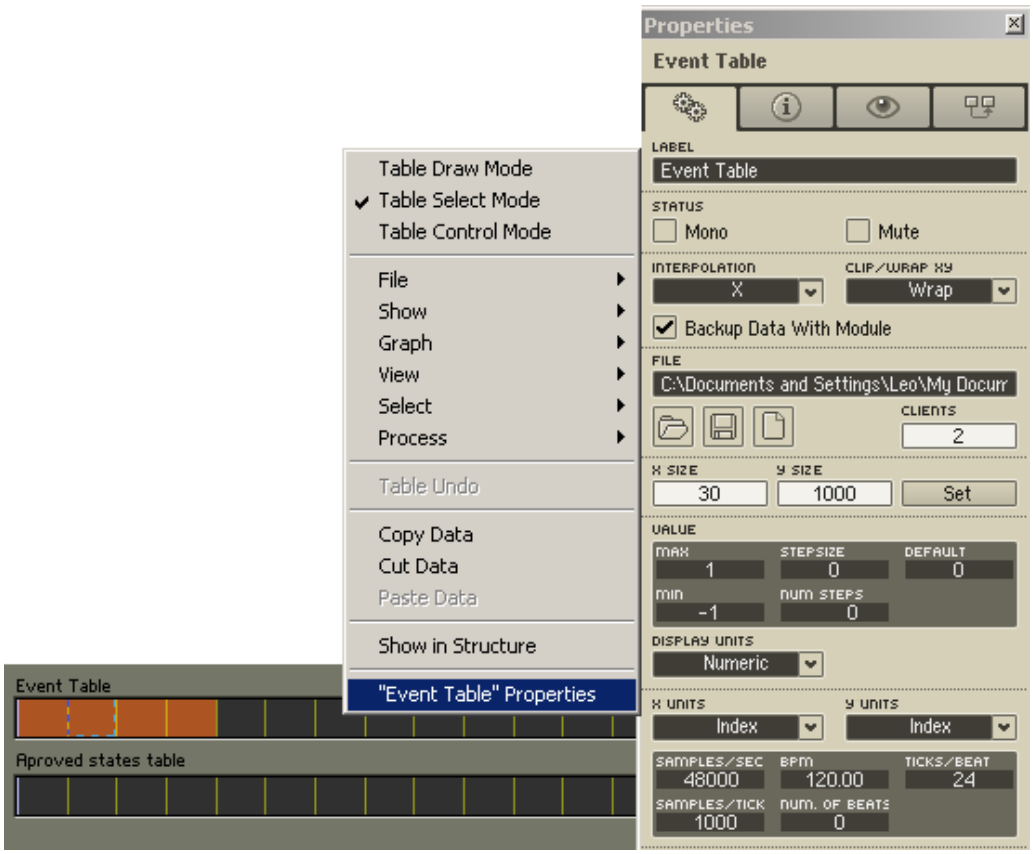


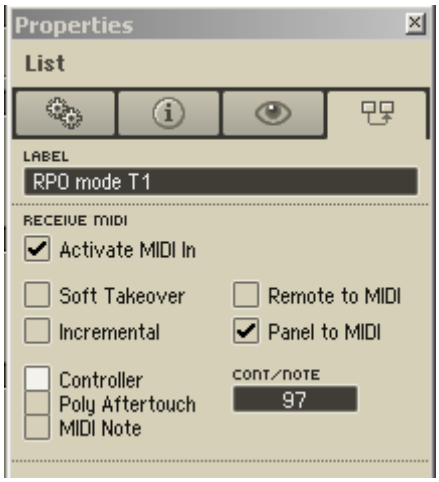

**Figure 10** Properties of the event table

Right click again in the event table but this time, select "Show in structure" and you will see a connections diagram. Double click on the "next scene prediction" block and then double click again the "Event table" block. You will see the same properties window. This time click on the button and load the same data as the one you stored just before. This will allow two event table blocks to share the data of the same event tables. If this procedure has being done correctly you should see that "Clients=2".

# 10 Adding new effects to the audio tracks

To add a new effect to the *ensemble* you have to:

- Route the audio lines, from the input, through the effect and then to the input of the audio table.
- Select the parameters you want to configure different for each scene.
- Go to the properties of each controller and clicking on the icon activate the MIDI parameters and assign a controller note according to the "MIDI assignations for the Scene memory" shown in the appendix of the manual.




Connect the effect's internal MIDI line to the Scene navigator's MIDI. This is done by selecting in the upper bar of the instrument, the Scene navigator's internal MIDI connectionas it is shown in the picture below for the case of the mixer.

**Figure 11** MIDI configurations

# 11 Appendix

## 11.1 Technical information about the program structure

## 11.2 MIDI assignations for the scene memory

To store the values of all the parameters in the panel in the “Scene memory” of the program there must be assigned, for each control, a component in the MIDI vector of the instrument.

To add an effect or other instrument to the program (and be able to configure it differently for every Scene), you will need to assign each parameter to the MIDI vector. The following table contains all the components used in the standard application, as well as the empty ones that are left out for your personal assignation.

(Unfinished version)

| Control # P.Prop | Voice Allocation | Parameter | Activate MIDI in | |
|---|---|---|---|---|
| X | 99 | Planned Scenes indicator | | |
| 97 | 98 | RPO mode T1.0 | X | |
| 96 | 97 | RPO mode T1.1 | X | |
| 95 | 96 | RPO mode T1.2 | X | |
| 94 | 95 | RPO mode T1.3 | X | |
| 93 | 94 | RPO mode T1.4 | X | |
| 92 | 93 | RPO mode T2.0 | X | |
| 91 | 92 | RPO mode T2.1 | X | |
| 90 | 91 | RPO mode T2.2 | X | |
| 89 | 90 | RPO mode T2.3 | X | |
| 88 | 89 | RPO mode T2.4 | X | |
| 87 | 88 | RPO mode T3.0 | X | |
| 86 | 87 | RPO mode T3.1 | X | |
| 85 | 86 | RPO mode T3.2 | X | |
| 84 | 85 | RPO mode T3.3 | X | |
| 83 | 84 | RPO mode T3.4 | X | |
| 82 | 83 | RPO mode T4.0 | X | |
| 81 | 82 | RPO mode T4.1 | X | |
| 80 | 81 | RPO mode T4.2 | X | |
| 79 | 80 | RPO mode T4.3 | X | |
| 78 | 79 | RPO mode T4.4 | X | |
| 77 | 78 | Monitor Track 1 | X | |
| 76 | 77 | Monitor Track 2 | X | |
| 75 | 76 | Monitor Track 3 | X | |
| 74 | 75 | Monitor Track 4 | X | |
| 73 | 74 | Scene origin selector T1 | X | |
| 72 | 73 | Scene origin selector T2 | X | |
| 71 | 72 | Scene origin selector T3 | X | |
| 70 | 71 | Scene origin selector T4 | X | |

| | | | | |
|---|---|---|---|---|
| 69 | 70 | Transfers yes? Track 1 | X | |
| 68 | 69 | Transfers yes? Track 2 | X | |
| 67 | 68 | Transfers yes? Track 3 | X | |
| 66 | 67 | Transfers yes? Track 4 | X | |
| 65 | 66 | | | |
| 64 | 65 | | | |
| 63 | 64 | | | |
| 62 | 63 | NEXT button (pedal) | X | |
| 61 | 62 | PREV button (pedal) | X | |
| 60 | 61 | RPO button (pedal) | X | |
| 59 | 60 | Mute Track 1 | X | |
| 58 | 59 | Mute Track 2 | X | |
| 57 | 58 | Mute Track 3 | X | |
| 56 | 57 | Mute Track 4 | X | |
| 55 | 56 | Punch in Track 1 | X | |
| 54 | 55 | Punch in Track 2 | X | |
| 53 | 54 | Punch in Track 3 | X | |
| 52 | 53 | Punch in Track 4 | X | |
| 51 | 52 | Input gain track 1 | X | |
| 50 | 51 | Input gain track 2 | X | |
| 49 | 50 | Input gain track 3 | X | |
| 48 | 49 | Input gain track 4 | X | |
| 47 | 48 | Feedback gain Track 1 | X | |
| 46 | 47 | Feedback gain Track 2 | X | |
| 45 | 46 | Feedback gain Track 3 | X | |
| 44 | 45 | Feedback gain Track 4 | X | |
| 43 | 44 | Track 1 Mixer level | | |
| 42 | 43 | Track 2 Mixer level | | |
| 41 | 42 | Track 3 Mixer level | | |
| 40 | 41 | Track 4 Mixer level | | |
| 39 | 40 | | | |
| 38 | 39 | | | |
| 37 | 38 | | | |
| 36 | 37 | | | |
| 35 | 36 | | | |
| 34 | 35 | | | |
| 33 | 34 | | | |
| 32 | 33 | | | |
| 31 | 32 | | | |
| 30 | 31 | | | |
| 29 | 30 | | | |
| 28 | 29 | | | |
| 27 | 28 | | | |
| 26 | 27 | | | |
| 25 | 26 | | | |
| 24 | 25 | | | |
| 23 | 24 | | | |
| 22 | 23 | | | |
| 21 | 22 | | | |
| 20 | 21 | | | |

| | | | | |
|---|---|---|---|---|
| 19 | 20 | | | |
| 18 | 19 | | | |
| 17 | 18 | | | |
| 16 | 17 | | | |
| 15 | 16 | | | |
| 14 | 15 | | | |
| 13 | 14 | | | |
| 12 | 13 | | | |
| 11 | 12 | | | |
| 10 | 11 | | | |
| 9 | 10 | | | |
| 8 | 9 | | | |
| 7 | 8 | | | |
| 6 | 7 | | | |
| 5 | 6 | | | |
| 4 | 5 | | | |
| 3 | 4 | | | |
| 2 | 3 | | | |
| 1 | 2 | | | |
| 0 | 1 | | | |
| | 0 | | | |

## 11.3 Terminology

*Template*
A form that contains all the configurations of the program for each scene. Represents the result of the programming stage of the software and it is used as the framework for the performance.

*Scenes*
Scenes are time intervals where the program has a fixed configuration and the musician is performing with the same instruments.

*Clips*
A clip is the audio contained in one track of one scene. All the *clips* in a scene represent all the audio contained in that scene.

*Overdub*
*Overdub* is to record over a previous existing file. The audio is superimposed and therefore the resulting audio file can not be separated into its components.

*Scroll bar*
It's an element in the front panel that allows the user to select a scene from the song.

*Clients*
When two or more elements of the program are accessing data from the same table, they are called *clients*.

*Ensemble*
A combination of instruments build in Reaktor is called an *ensemble*. Scene builder is an *ensemble*.

*RPO evolution:*
Within each scene, a track can be performing in different modes: recording; overdubbing, playing, transferring audio or doing nothing. The *RPO evolution* is a sequence of these internal modes of the Scene. You can select one of the modes from a list, and then by pressing the RPO button in the pedal you will switch between them in order to create the clip for your scene.

*Macros:*
Macro is a generic name for structures or building blocks made in Reaktor. Most of the *macros* of Scene builder are made from using the core structure (lower level of this programming language)